Computational Geometry 43 (2010) 331–356

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

The complexity of flow on fat terrains and its I/O-efficient computation

Mark de Berg [a,1], Otfried Cheong [b,2], Herman Haverkort [a,*], Jung-Gun Lim [b], Laura Toma [c,3]

[a] Dept. of Computer Science, Eindhoven University of Technology, The Netherlands
[b] Korea Advanced Institute of Science and Technology, Republic of Korea
[c] Bowdoin College, USA

ARTICLE INFO

Article history:
Received 4 July 2007
Received in revised form 29 October 2008
Accepted 8 December 2008
Available online 21 June 2009

Keywords:
River network
Watershed
Polyhedral terrain
Fatness
Realistic input models

ABSTRACT

We study the complexity and the I/O-efficient computation of flow on triangulated terrains. We present an acyclic graph, the descent graph, that enables us to trace flow paths in triangulations I/O-efficiently. We use the descent graph to obtain I/O-efficient algorithms for computing river networks and watershed-area maps in $O(Sort(d+r))$ I/O's, where r is the complexity of the river network and d of the descent graph. Furthermore we describe a data structure based on the subdivision of the terrain induced by the edges of the triangulation and paths of steepest ascent and descent from its vertices. This data structure can be used to report the boundary of the watershed of a query point q or the flow path from q in $O(l(s)+Scan(k))$ I/O's, where s is the complexity of the subdivision underlying the data structure, $l(s)$ is the number of I/O's used for planar point location in this subdivision, and k is the size of the reported output.

On *α-fat terrains*, that is, triangulated terrains where the minimum angle of any triangle is bounded from below by α, we show that the worst-case complexity of the descent graph and of any path of steepest descent is $O(n/\alpha^2)$, where n is the number of triangles in the terrain. The worst-case complexity of the river network and the above-mentioned data structure on such terrains is $O(n^2/\alpha^2)$. When α is a positive constant this improves the corresponding bounds for arbitrary terrains by a linear factor. We prove that similar bounds cannot be proven for Delaunay triangulations: these can have river networks of complexity $\Theta(n^3)$.

1. Introduction

In this paper we study one of the most important problems on terrains: analyzing the flow of water. The basic questions in flow analysis are to identify the river network and, for a given point q, its watershed (the part of the terrain from which water flows to q). Acquiring real flow data for a terrain is tedious, time-consuming and, for large terrains, often impossible. Fortunately, terrain elevation data is now widely available at high resolution. As a result, flow modeling and analysis based on elevation data is a popular topic for researchers in GIS (geographic information systems) and standard GIS packages provide some form of flow analysis.

* Corresponding author.
E-mail addresses: mdberg@win.tue.nl (M. de Berg), otfried@kaist.edu (O. Cheong), cs.herman@haverkort.net (H. Haverkort), araste@gmail.com (J.-G. Lim), ltoma@bowdoin.edu (L. Toma).

[1] M.d.B. was supported by the Netherlands' Organisation for Scientific Research (NWO) under project No. 639.023.301.
[2] O.C. was supported by the Korea Science and Engineering Foundation Grant R01-2008-000-11607-0 funded by the Korean government.
[3] L.T. was supported by National Science Foundation award No. 0728780.

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2008.12.008

One common representation of terrain data in a GIS is the TIN (triangulated irregular network). A TIN—in computational-geometry terms: a triangulated polyhedral terrain—is obtained by triangulating a collection of irregularly spaced sample points and then giving each triangulation vertex the elevation of the corresponding sample point. When working with very large terrains, the data is too large to fit into the computer's main memory. Most of the data will therefore have to reside on disk during the computation, making I/O (moving data between main memory and disk) the bottleneck of the computation. This leads us to the topic of our paper: the study of river networks and watersheds on TINs, and the design of I/O-efficient algorithms for computing these structures.

We analyze our algorithms with the model introduced by Aggarwal and Vitter [3], which has become the standard model for I/O-efficient algorithms. In this model, a computer has an internal memory of size M and an arbitrarily large external memory (disk) where data is stored in blocks of size B. Whenever an algorithm wants to work on data not present in internal memory, the block(s) containing those data are read from external memory. Writing data to external memory is also done in blocks. The I/O-*complexity* of an algorithm in this model is measured in terms of the number of I/O's—reading or writing a block from or to external memory—it performs. In this model, scanning (reading a set of n consecutive items from disk) takes $Scan(n) = \Theta(n/B)$ I/O's, and sorting $Sort(n) = \Theta((n/B)\log_{M/B}(n/B))$ I/O's in the worst case.

1.1. Related work

The previous work on modeling flow on TINs falls into two classes. Most GIS papers [14,20–23] adopt a *discrete* approach and route flow from a triangle to one of its three neighbor triangles using the direction of steepest descent, for example from the center of the triangle. This approach is appealing because of its simplicity; it is problematic, however, because it discretizes flow and tends to lead to inconsistencies when the triangles in the TIN differ a lot in size [24,25].

The approach taken in the computational-geometry literature considers the TIN as a *continuous* surface on which water always flows in the direction of steepest descent. De Berg et al. [11], McAllister [16,17], McAllister and Snoeyink [18] and Yu et al. [25] study the structure and the complexity of the river network and other structures on TIN sunder this model. In particular, de Berg et al. [11] prove that the complexity of the river network—see Section 2 for a formal definition—on a TIN of n vertices can be $\Theta(n^3)$ in the worst case: there can be $\Theta(n)$ separate rivers that each have complexity $\Theta(n^2)$. McAllister [16,17] presents an algorithm to compute the boundaries between the watersheds of the local minima in the terrain. None of the papers mentioned above provide I/O-efficient algorithms.

I/O-efficient flow modeling was first studied by Arge et al. [4]. They, however, consider grids, the other common type of data representation in GIS. Their system, *Terraflow*, has become the state of the art in flow modeling on very large grids. *Terraflow* uses a discrete approach which can easily be extended to TINs. However, discrete flow is only an approximation of real flow. Ever since *Terraflow* appeared, the challenge is to refine the approach and develop I/O-efficient algorithms to model continuous flow on TINs, which is ultimately more accurate.

The main step in the computation of flow, and at the same time the bottleneck in the I/O-model, is tracing paths of steepest descent across the triangles that they intersect. In particular, any river is such a path of steepest descent. While in internal memory a path of size k can be traced in $O(k)$ time, the best known I/O-bound is $O(k/\log B)$ I/O's on planar graphs [1]. This results in an upper bound of at least $O(n/B + r/\log B)$ on the number of I/O's needed to compute a river network of size r, but that many I/O's would be prohibitively expensive.

Moreover, de Berg et al. [11] showed that r is $\Theta(n^3)$ in the worst case. However, the worst-case example in their proof is sufficiently contrived that such inputs are unlikely to occur in real life. In computational geometry such discrepancies between worst case and practice have led to the study of input models that resemble realistic inputs better. Moet et al. [19] studied visibility and distance problems on *realistic terrains*. In this paper we consider flow modeling on *fat terrains*, that is, terrains where the minimum angle of any triangle is bounded from below by a positive constant.[4] Our notion of a fat terrain is less restrictive than the notion of realistic terrains from Moet et al.

1.2. Our results

In this paper we give improved bounds for the complexity of the river network on a terrain which is fat or has no obtuse triangles, and show how to compute a number of flow-related structures I/O-efficiently. The main ingredient in our solution is to represent the terrain by a directed graph, which we call the *descent graph*, $\mathcal{G}_{\text{desc}}$. The nodes of $\mathcal{G}_{\text{desc}}$ represent the edges of the triangulation, and we define the arcs of $\mathcal{G}_{\text{desc}}$ such that following a path of steepest descent on the terrain corresponds to following a path in $\mathcal{G}_{\text{desc}}$. Unfortunately, in its basic form, $\mathcal{G}_{\text{desc}}$ can have cycles, and a path of steepest descent can visit the same edge more than once. This is, in fact, exactly the reason why the complexity of a path of steepest descent in an arbitrary terrain can be $\Theta(n^2)$: it can visit a linear number of edges each a linear number of times [11]. In the I/O-model, a descent graph with cycles does not only signify a potentially problematic output size, but it also constitutes an algorithmic problem, because it is not known how to store such a graph on disk such that any path of length k can be traced using $O(k/B)$ I/O's [1].

[4] Here the angle of a triangle is measured in space, that is, in the plane containing the triangle. Our results also hold if the angles are measured in the projection on the xy-plane, or if we require that all triangles be non-obtuse.

The cornerstone of this paper is an idea that solves both of these problems at the same time: we subdivide each edge of the triangulation into a number of segments, in such a way that the descent graph, when defined on these segments instead of the original edges, is *acyclic*. Moreover, we show that for fat terrains a constant number of segments per edge suffices. This implies that any path of steepest descent can visit each segment at most once, and hence its worst-case complexity is $\Theta(n)$. This improves the $\Theta(n^2)$ bound for general (arbitrary) terrains by a linear factor.

The $O(n)$ bound on the complexity of a single path of steepest descent implies an $O(n^2)$ bound on the complexity s of the *strip map*: the subdivision $\mathcal{S}$ of the terrain induced by the paths of steepest ascent and descent from all vertices. It also follows that the complexity r of the river network on a fat terrain is $O(n^2)$, which is again a linear factor smaller than the $O(n^3)$ bound for river networks on general terrains.

We show that for terrains without obtuse triangles we have the same bounds on the size of paths of steepest descent, the strip map and the river network as for fat terrains. We also describe how to construct a fat terrain of n non-obtuse triangles where the river network actually has complexity $\Omega(n^2)$. This shows that our bounds are tight in the worst case.

The acyclicity of the descent graph allows us to apply time-forward processing, a standard technique for processing directed acyclic graphs I/O-efficiently [6,9], traversing the paths in a batched I/O-efficient manner. By applying and refining ideas from McAllister [16] and Yu et al. [25], we obtain the following algorithms and data structures, all computable in $O(Sort(s))$ I/O's:

- an algorithm to compute the river network, with a piecewise quadratic function of $O(r+n)$ pieces whose value is the area of the watershed for each point of the network;
- a data structure that reports the boundary of the watershed of any query point q (the region of the terrain from which water flows to q) in $O(l(s)+k/B)$ I/O's, where $l(s)$ is the number of I/O's needed to locate q in the strip map $\mathcal{S}$ of size s, and k is the complexity of the reported watershed;
- a data structure that reports the flow path from any query point q (the course of water flowing from q) in $O(l(s)+k/B)$ I/O's, where $l(s)$ is as defined above and k is the complexity of the reported path.

Note that the current best-known bound for $l(s)$ is $O(\log_B^2 s)$ when we require that the necessary point location structure on $\mathcal{S}$ is built in $O((s/B)\log_B s)$ I/O's [5]; a bound of $O(\log_B s)$ on $l(s)$ is possible but known data structures that achieve that bound cannot be built I/O-efficiently [7,12].

One of the open questions posed by de Berg et al. [11] was whether one could prove an $O(n^2)$ bound on the complexity of river networks in Delaunay triangulations. We answer this question negatively and show how to construct a Delaunay triangulation of n triangles with a river network of size $\Theta(n^3)$.

1.3. Outline

The rest of the paper is organized as follows. Section 2 reviews basic terminology and properties of terrains that are used in this paper. Section 3 introduces the descent graph, and defines an acyclic descent graph with only $O(n)$ nodes for fat terrains and for terrains without obtuse triangles. Thus we prove an $O(n^2)$ bound on the complexity of the strip map and the river network in these cases. Section 4 shows how to use the descent graph to compute the river network and the watershed-area map I/O-efficiently, and Section 5 describes our data structure for I/O-efficient flow path and watershed boundary queries. In Section 6 we briefly discuss how to deal with deviations from the terrain properties that we assumed in Section 2. Section 7 describes how to construct a terrain of n fat, non-obtuse triangles with a river network of complexity $\Theta(n^2)$ (thus proving that the bounds from Section 3 are tight), and how to construct a Delaunay-triangulated terrain of n triangles whose river network has complexity $\Theta(n^3)$. We conclude and comment on our results in Section 8.

2. Preliminaries

2.1. The terrain

Let $\mathcal{T}$ be a TIN defined on n vertices. To model flow we assume that water always runs downhill in the direction of steepest descent. Furthermore we assume that the direction of steepest descent is unique for any point on the terrain (which implies that there are no horizontal triangles in $\mathcal{T}$), no water reaches the boundary of the terrain, and no edge is parallel to the direction of steepest descent on an adjacent triangle. We discuss how to do without these last three assumptions in Section 6.

2.2. Classification of edges

Following Yu et al. [25], we distinguish three types of edges in $\mathcal{T}$, depending on whether the direction of steepest descent on the adjacent triangles is directed towards or away from the edge: *transfluent edges* are edges that receive water from one adjacent triangle, which continues its way down the other triangle; *channels* are edges that receive water from both adjacent triangles; *ridges* are edges that do not receive water from any adjacent triangle. Since we assume that the direction of steepest descent is unique everywhere, there cannot be horizontal channels, and since no water reaches the boundary of the terrain, all edges on the boundary of the terrain are ridges.

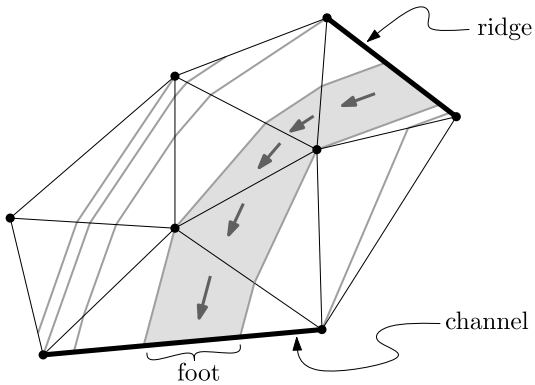

Fig. 1. A part of a terrain and a strip (in light gray) on it. Up- and down-paths are shown in dark gray; the arrows indicate the direction of flow within the strip.

2.3. Slope profiles

The direction of steepest descent from a vertex may be along an incident edge, or on an incident triangle orthogonal to its contour lines. To capture this, we define the *slope profile*[5] of a point p on the terrain as the function $s_p : S^1 \to \mathbb{R}$ such that $s_p(\theta)$ is the slope of the path that leaves p in the direction θ. The interesting directions for a vertex v of $\mathcal{T}$ correspond to the local maxima and minima in the slope profile of v. Note that these directions include the directions of channels and ridges incident to v, and also the directions of steepest descent and ascent. A vertex v is a *pit* if its slope profile is entirely positive.

2.4. Up-paths, down-paths, and the strip-map

We define up-paths and down-paths as paths of locally steepest ascent or descent as follows. An *up-path* from p is a path that starts at p, goes into a direction θ that is a positive local maximum in the slope profile of p leading onto the interior of an incident triangle, and then follows the steepest ascent until a vertex or a ridge of $\mathcal{T}$ is reached. Note that up-paths never run along edges, since they cannot enter channels, they end as soon as a ridge is reached, and when a transfluent edge is reached, the path continues across the interior of the triangle above that edge, not along the edge. Similarly, a *down-path* from p is a path that starts at p, goes into a direction θ that is a negative local minimum in the slope profile of p leading onto the interior of an incident triangle, and then follows the steepest descent until a vertex or a channel of $\mathcal{T}$ is reached. Like up-paths, down-paths never run along edges. Therefore both up-paths and down-paths consist exclusively of segments across the interior of triangles. These segments are orthogonal to the contour lines of the triangles. The beginning and the end of an up-path or down-path may be vertices of the triangulation. The remaining endpoints of the segments that constitute the path must lie in the interior of transfluent edges. In particular, if p is a vertex of $\mathcal{T}$ or a point in a channel, then each segment of an up-path from p to some point q is traversed in the direction of steepest ascent of its triangle; if we traverse the segments in opposite direction then we follow the direction of steepest descent and have a down-path from q to p. Therefore, in the rest of this paper we may sometimes refer to a down-path as an up-path or the other way around, depending on our point of view.

The *strip map* $\mathcal{S}$ of $\mathcal{T}$ is the subdivision of the terrain induced by the channels, the ridges, and the up-paths and down-paths from all vertices of $\mathcal{T}$. The $O(n)$ faces of the strip map are called *strips*. Each strip is bounded by a portion of a ridge, a portion of a channel (the foot), and two possibly empty chains of up-paths. Note that from every point at the foot of a strip, the up-path through the strip has the same combinatorial structure: it crosses the same triangles and leads to the same ridge (see Fig. 1).

2.5. Watersheds and the river network

The *watershed* $W(q)$ of a point q on the terrain is the set of all points on $\mathcal{T}$ from which the water flows to q. The *river network* of $\mathcal{T}$ is the set of all points on $\mathcal{T}$ with watersheds of non-zero area, or in other words, the set of points whose watersheds are two-dimensional regions. De Berg et al. [11] argue that the river network of $\mathcal{T}$ consists of the channels of $\mathcal{T}$ and of the paths of steepest descent that start from channels. Any path of steepest descent that starts on a channel e consists of a section of e and the steepest-descent path leaving from the lower endpoint of e. Hence we can define the river

[5] McAllister works with the *height profile*, which is a function that gives the elevation of points around p at a small distance ϵ from p. For our purposes, the slope profile and the height profile functions are equivalent: they have the same global and local extrema.

network more precisely as the union of the channels of $\mathcal{T}$ and the paths of steepest descent from the lower endpoints of the channels down to the point where they reach another channel or a pit.

3. Modeling the terrain with a descent graph

In this section we describe how to model a triangulated terrain $\mathcal{T}$ using a *descent graph*, $\mathcal{G}_{\text{desc}}$. To define $\mathcal{G}_{\text{desc}}$, assume that the transfluent edges of $\mathcal{T}$ have been subdivided into *segments*, as will be described below; each ridge or channel is one segment. Then $\mathcal{G}_{\text{desc}}$ contains a node for each vertex of $\mathcal{T}$ and for each segment of $\mathcal{T}$; let $seg(v)$ denote the vertex/segment corresponding to node v. To ensure that the segments $seg(v)$, when viewed as sets of points in the plane, are disjoint, we consider a segment to include its upper endpoint (unless this is a vertex of $\mathcal{T}$), and to exclude its lower endpoint (ties are broken arbitrarily, but in a consistent way along a single edge of the terrain). $\mathcal{G}_{\text{desc}}$ contains a directed arc from node a to node b if and only if one of the following applies: (1) $seg(a)$ is the upper endpoint of $seg(b)$, and $seg(b)$ is a channel; (2) $seg(b)$ is the lower endpoint of $seg(a)$, and $seg(a)$ is a channel; (3) $seg(a)$ and $seg(b)$ are on the boundary of the same triangle Δ, and there is a path of steepest descent across the interior of Δ from $seg(a)$ to $seg(b)$. It can be seen that any path of steepest descent in $\mathcal{T}$ corresponds to a path in $\mathcal{G}_{\text{desc}}$. Our goal is now to ensure that $\mathcal{G}_{\text{desc}}$ is small and acyclic.

In Section 3.1 we describe a set of requirements on the subdivision of the edges of $\mathcal{T}$ which imply that $\mathcal{G}_{\text{desc}}$ is acyclic; we call a subdivision satisfying these requirements *compliant*. For α-fat terrains, that is, terrains where the minimum angle of each triangle is at least $\alpha > 0$, we show how to construct a compliant subdivision of size $O(n/\alpha^2)$ in Section 3.2. This leads to a bound of $O(n^2/\alpha^2)$ on the complexity of the river network and the strip map of such terrains, whereas general terrains can have a river network with complexity $\Theta(n^3)$.

In Section 3.3, we prove that if the terrain does not contain any obtuse triangles (triangles with an angle of more than 90 degrees), the descent graph is acyclic even if it is based directly on the vertices and the original, undivided edges of the triangulation. Thus the river network and the strip map of a non-obtuse terrain have complexity $O(n^2)$.

Note that in the rest of this section the angles of a triangle are always measured in the plane that contains the triangle. However, the reader may verify that the definitions and arguments presented in this section can also be applied to the projection of the terrain on a horizontal plane: any reference to triangles, vertices, edges, segments thereof, contour lines, lines of steepest descent, etc. should then be understood as referring to objects in the horizontal plane; angles and distances are measured in the projection on that plane; enclosing balls become enclosing disks in the plane etc.—elevations only matter where explicitly mentioned. Thus the above asymptotic bounds also hold for terrains that are fat or non-obtuse in the sense that their *projections* on the horizontal plane consist of only fat triangles or only non-obtuse triangles. The constants hidden by the O-notation may be better in this case.

3.1. Compliant subdivision

Consider the following set of requirements on the subdivision of the edges of $\mathcal{T}$. We call a subdivision that meets these requirements *compliant*.

- Channels and ridges of $\mathcal{T}$ are not subdivided: each channel or ridge is one segment;
- For any segment s on a transfluent edge e and any triangle Δ incident to e, we have:
  - (i) If s is not incident to a vertex of Δ, then the (open) smallest enclosing ball of s (that is, the ball with s as a diameter) does not intersect any other edge of Δ;
  - (ii) If s is incident to a vertex p of Δ, and the (open) ball centered at p with s as a radius intersects another segment s' on an edge of Δ, then either s' is incident to p and $|s| = |s'|$, or s' is separated from s by a line of steepest descent through p on Δ.

Let us denote the endpoint of $seg(v)$ with highest elevation by $up(v)$, the endpoint with lowest elevation by $lw(v)$, and the midpoint of $seg(v)$ by $md(v)$. If $seg(v)$ is horizontal, the tie is broken arbitrarily (this cannot happen for channels). If $seg(v)$ is a vertex, we have $up(v) = md(v) = lw(v) = seg(v)$. The *anchor* $an(v)$ of v is defined as follows (see also Fig. 2):

- if $seg(v)$ is a channel, then $an(v) = lw(v)$;
- if $seg(v)$ is a vertex or a segment of a transfluent edge, then $an(v) = md(v)$;
- if $seg(v)$ is a ridge, then $an(v) = up(v)$.

We now study what it means for the elevation of the anchors if water can flow from $seg(a)$ to $seg(b)$. Let $z(p)$ denote the elevation of a point p on the terrain.

Lemma 1. *Given $\mathcal{G}_{\text{desc}}$ of a compliant subdivision, if $seg(a)$ and $seg(b)$ are on the boundary of the same triangle Δ, $seg(b)$ is not a channel, and there is a path of steepest descent across the interior of Δ from $seg(a)$ to $seg(b)$, then $z(up(a)) \geqslant z(up(b))$, and equality can hold only if $seg(a)$ is incident to $up(b)$.*

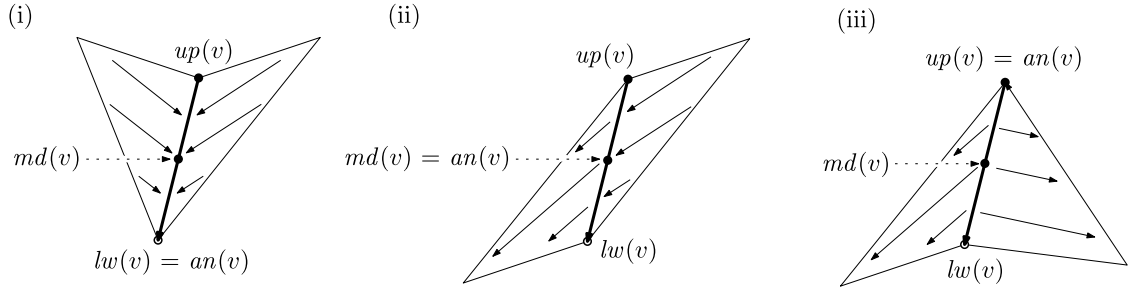

Fig. 2. The anchor for (i) a channel, (ii) a transfluent edge, (iii) a ridge. The arrows indicate the flow direction.

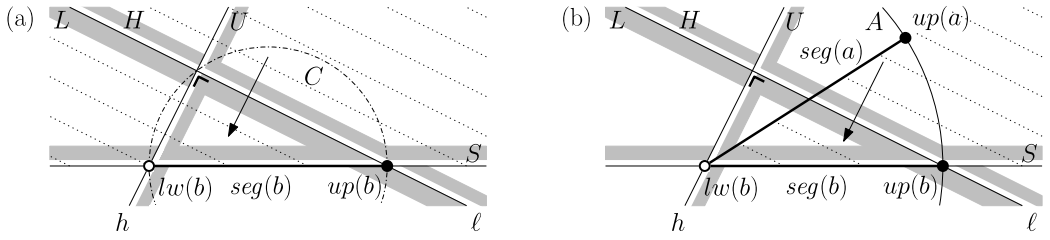

Fig. 3. Notation for the proof of Lemma 1.

Proof. Let $p \in seg(a)$ and $q \in seg(b)$ be such that the segment pq lies in the interior of Δ and follows the direction of steepest descent. If $seg(b)$ is a vertex of $\mathcal{T}$, then $z(up(a)) \geqslant z(p) > z(q) = z(up(b))$. Since $seg(b)$ is not a ridge or a channel, the only other case is that $seg(b)$ is a segment on a transfluent edge.

Consider now the plane Γ supporting Δ. Let S be the open half-plane on Γ containing p, bounded by the line through $seg(b)$. Let ℓ be the intersection of Γ with the horizontal plane through $up(b)$, and let h be the line on Γ orthogonal to ℓ through $lw(b)$. Note that ℓ is a contour line of Γ, the line h follows the direction of steepest descent on Γ, and pq is parallel to h.

Let L and H be the closed lower and the open upper half-planes of Γ bounded by ℓ, and let U be the open half-plane of Γ bounded by h containing $seg(b)$. Let C be the open disk on Γ bounded by the minimum circumscribed circle of $seg(b)$. See Fig. 3(a) for an illustration. Since $q \in seg(b)$, we must have $p \in U \cap S$. We distinguish three cases:

First, if $seg(b)$ is not incident to a vertex of $\mathcal{T}$, then by compliance condition (i) C does not intersect $seg(a)$. By Thales' theorem, the rectangular triangle $U \cap S \cap L$ lies in C, and so $p \notin L$. This implies $z(up(a)) \geqslant z(p) > z(up(b))$.

Second, if $lw(b)$ is a vertex of $\mathcal{T}$, then let A be the open disk on Γ centered at $lw(b)$ with $up(b)$ on its boundary. If $seg(a)$ is not incident to $lw(b)$, then by compliance condition (ii) it does not intersect A (since $p \in U \cap seg(a)$, $seg(a)$ is not separated from $seg(b)$ by h). Since $C \subset A$, we again have $z(up(a)) \geqslant z(p) > z(up(b))$. If $seg(a)$ is incident to $lw(b)$ (Fig. 3(b)), then by compliance condition (ii) $up(a)$ must lie in $U \setminus A$, and hence in H, implying $z(up(a)) > z(up(b))$.

Finally, if $up(b)$ is a vertex of $\mathcal{T}$, then let A' be the open disk on Γ centered at $up(b)$ with $lw(b)$ on its boundary. If $seg(a)$ is not incident to $up(b)$, then by compliance condition (ii) it does not intersect A', and p lies again in H. If $seg(a)$ is incident to $up(b)$, then clearly $z(up(a)) \geqslant z(up(b))$. □

Corollary 1. *If $seg(a)$ and $seg(b)$ are on the boundary of the same triangle Δ, $seg(a)$ is not a ridge, and there is a path of steepest descent across the interior of Δ from $seg(a)$ to $seg(b)$, then $z(lw(a)) \geqslant z(lw(b))$, and equality can only hold if $seg(b)$ is incident to $lw(a)$.*

Proof. We mirror the terrain in a horizontal plane, and apply Lemma 1. (Note that the proof of Lemma 1 did not exploit the special properties of our terrain that would not be preserved under the reflection.) □

We now prove that any descent graph $\mathcal{G}_{\text{desc}}$ based on a compliant subdivision is acyclic. To this end, we define a partial order $\succ$ as follows: For two nodes a and b in $\mathcal{G}_{\text{desc}}$, we define $a \succ b$ if and only if $z(an(a)) > z(an(b))$, or if $z(an(a)) = z(an(b))$ and a is a channel and b is a vertex. Clearly $\succ$ is indeed a partial order.

Lemma 2. *Given $\mathcal{G}_{\text{desc}}$ of a compliant subdivision, if $\mathcal{G}_{\text{desc}}$ contains an arc from a to b, then $a \succ b$.*

Proof. If $seg(a)$ is the upper endpoint of the channel $seg(b)$, then $z(an(b)) = z(lw(b)) < z(up(b)) = z(an(a))$. If $seg(b)$ is the lower endpoint of the channel $seg(a)$, then $z(an(b)) = z(lw(a)) = z(an(a))$. Since a is a channel and b is a vertex, we have $a \succ b$.

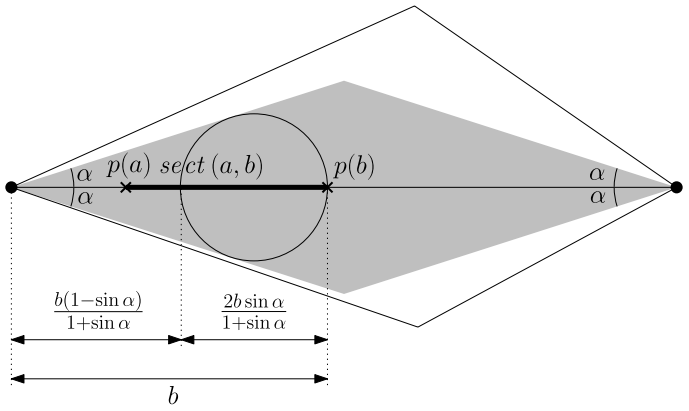

Fig. 4. Placing a ball on $sect(a,b)$ that does not intersect any other segments or edges.

It remains to discuss the case where there is a path of steepest descent from some point $p \in seg(a)$ to some point $q \in seg(b)$ across the interior of their common triangle Δ. We note that $seg(a)$ cannot be a channel, and $seg(b)$ cannot be a ridge. We have $z(up(a)) \geqslant z(p) > z(q) \geqslant z(lw(b))$. We now distinguish four cases, proving $z(an(a)) > z(an(b))$ and therefore $a \succ b$ in each case:

First, if $seg(a)$ is a ridge and $seg(b)$ is a channel, then $z(an(a)) = z(up(a)) > z(lw(b)) = z(an(b))$.

Second, if $seg(a)$ is a ridge and $seg(b)$ is a vertex or a segment of a transfluent edge, then $z(an(a)) = z(up(a)) \geqslant z(up(b))$ (by Lemma 1). Since $z(up(a)) > z(lw(b))$, then $z(an(a)) > z(md(b)) = z(an(b))$.

The case where $seg(a)$ is a vertex or a segment of a transfluent edge and $seg(b)$ is a channel is handled symmetrically with Corollary 1.

Finally, if both $seg(a)$ and $seg(b)$ are vertices of the triangulation or segments of transfluent edges, then we have both $z(up(a)) \geqslant z(up(b))$ and $z(lw(a)) \geqslant z(lw(b))$ by Lemma 1 and Corollary 1. We cannot have equality in *both* inequalities, since if $seg(a)$ and $seg(b)$ shared both $up(b)$ and $lw(a)$, that would imply $up(b) = lw(a)$ and thus $z(up(a)) = z(up(b)) = z(lw(a)) = z(lw(b))$, which contradicts $z(p) > z(q)$. It follows that we have equality in at most one of the two inequalities, so $z(an(a)) = z(md(a)) > z(md(b)) = z(an(b))$. □

Corollary 2. *$\mathcal{G}_{\text{desc}}$ of a compliant subdivision is a planar directed acyclic graph.*

Proof. Since $\succ$ is a partial order, $\mathcal{G}_{\text{desc}}$ is acyclic. A planar straight-line embedding is given by placing every node v at the projection of $md(v)$ on a horizontal plane. □

3.2. Linear size compliant subdivisions for fat terrains

In this section we show that we can build a compliant subdivision of size $O(n)$ (and therefore a descent graph of size $O(n)$) if the terrain $\mathcal{T}$ is *fat*, that is, if the minimum angle of each triangle in $\mathcal{T}$ is bounded from below by a positive constant α.

The main idea of our approach is the following. We cut every transfluent edge into three pieces, such that the two pieces that are incident to the endpoints of the edge—we call them the *heads*—satisfy compliance condition (ii). The remaining piece in the middle—the *body*—is then subdivided further so that the resulting subdivision satisfies condition (i). We first explain the subdivision of the body.

Lemma 3. *Let s be the segment obtained by removing at least a fraction ρ at the two ends of an edge e of $\mathcal{T}$. Then s can be subdivided into at most $2\lceil \frac{1}{2\alpha} \ln \frac{1}{2\rho} \rceil$ segments fulfilling compliance condition* (i)*.*

Proof. We cut s in two parts at the midpoint of e, and define segments for each part separately. Refer to Fig. 4. For any of the two parts, let $p(x)$ be the point on s at distance x from the nearest endpoint of e. Let $sect(a,b)$ be the segment between $p(a)$ and $p(b)$, and let $N(a,b)$ be the number of subsegments needed to cover it with segments that fulfill compliance condition (i). Note that at point $p(x)$, the distance to any other edge of the two triangles incident to e is at least $x \sin\alpha$. We can therefore put a ball at $p(b/(1+\sin\alpha))$ with radius $(b\sin\alpha)/(1+\sin\alpha)$, which covers $sect(b(1-\sin\alpha)/(1+\sin\alpha), b)$ and does not intersect any other edge. This implies

$$N(a,b) \leqslant 1 + N\big(a, b(1-\sin\alpha)/(1+\sin\alpha)\big).$$

With base case $N(a,b) = 0$ for $b \leqslant a$, we get $N(a,b) \leqslant \lceil \ln(b/a)/\ln((1+\sin\alpha)/(1-\sin\alpha)) \rceil$. Since $\ln(b/a) < \ln\frac{1}{2\rho}$ and $\ln((1+\sin\alpha)/(1-\sin\alpha)) > 2\alpha$, the lemma follows. □

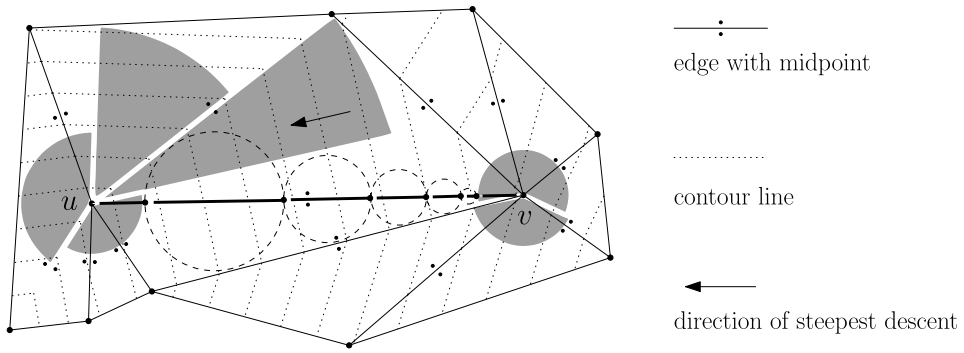

Fig. 5. An example of the subdivision of edges in a terrain. To illustrate the concepts most clearly, we projected the terrain on a horizontal plane and measured distances and angles in the plane instead of in three-dimensional space. The shaded regions are the neighborhoods around u and v. Note that the two large neighborhoods to the upper right of u are in fact empty: no edges of the triangulation intersect their interiors. For the body of the edge between u and v, a subdivision into segments fulfilling compliance condition (i) is shown.

If the maximum degree of the vertices in the triangulation is at most d, we can now get a bound on the total number of segments per transfluent edge as follows. For every vertex p of $\mathcal{T}$, let $r(p)$ be the distance from p to the nearest edge not incident to p, and we create a head segment of length $r(p)/2$ on each transfluent edge incident to p. Doing this for every vertex of $\mathcal{T}$, we create two heads on every transfluent edge. We subdivide the bodies of the transfluent edges with the technique of Lemma 3.

Lemma 4. *The above algorithm results in a compliant subdivision where each transfluent edge is divided into two heads and at most* $2\lceil \frac{d}{2\alpha} \ln \frac{1}{\sin\alpha} \rceil$ *body segments.*

Proof. Consider the edges incident to a single vertex p of $\mathcal{T}$, in clockwise order around p. Since the minimum angle in any triangle is α, the length ratio of any two consecutive edges in the clockwise order is at least $\sin\alpha$, and the length ratio between any two edges incident to p is at least $(\sin\alpha)^{d-1}$. Furthermore, for any triangle Δ incident to p, the distance from p to the opposite edge of Δ is at least $\sin\alpha$ times the length of any edge of Δ incident to p. Hence $r(p)$ is at least $(\sin\alpha)^d$ times the length of any edge (of any triangle) incident to p, and we can apply Lemma 3 with $\rho = \frac{1}{2}(\sin\alpha)^d$. □

Note that if α were a lower bound on the minimum angle of any triangle measured in the projection on the xy-plane, we would have $d \leqslant 2\pi/\alpha$. Thus, we would get a constant number of segments per edge and a descent graph of linear size when α is constant. However, when α is a lower bound on the angle in any triangle only when the angles are measured in space, d can be arbitrarily large: any number of almost vertical fat triangles could meet in a vertex. Below we describe a more refined technique to define the heads. With this technique we will create larger heads and therefore less body segments, leading to smaller descent graphs in practice and eliminating the dependency on the vertex degree in the theoretical bound.

We consider each edge of the terrain $\mathcal{T}$ to consist of two half-edges, separated by the midpoint of the edge. A vertex *owns* the half-edges incident to it. For a vertex v in the triangulation, let $\theta_0, \ldots, \theta_{k-1}$ be the k local extrema in the slope profile of v, in counterclockwise order around v. We define $\theta_k := \theta_0$. For $i \in \{0, \ldots, k-1\}$, the *sector* S_i of v consists of all points whose projections on the horizontal plane through v lie in the open wedge W_i that lies to the left of the halfline that extends from v in the direction θ_i and to the right of the halfline that extends from v in the direction θ_{i+1}. Let r_i be the distance from v to the nearest intersection of S_i with a half-edge not owned by v. The neighborhood V_i of v is now defined as the set of points in S_i that lie at a distance less than r_i from v—it is bounded by the vertical planes that contain the halflines from v in the directions θ_i and θ_{i+1}, and by the surface of the ball centered at v with radius r_i. The neighborhood boundaries divide every transfluent edge in the triangulation into two *heads* (the segments inside the neighborhoods of the endpoints) and at most one *body* (the rest of the edge, if not empty)—see Fig. 5. No point on an edge of the triangulation lies in the interior of more than one neighborhood: points on transfluent half-edges lie in a single neighborhood of its owner, while ridges and channels constitute local extrema in the slope profiles of their endpoints and therefore do not lie in the interior of any neighborhoods.

We further subdivide the bodies of transfluent edges into a minimum number of segments fulfilling compliancy condition (i) (Lemma 3 gives an upper bound on the number of segments needed)—see Fig. 5 for an example.

Lemma 5. *If the angles of all triangles are bounded from below by* $\alpha > 0$*, then each head covers at least a fraction* $1/2^{1+3\pi/\alpha}$ *of its edge.*

Proof. Consider a sector S_i of a vertex v. The *spokes* of S_i are the following line segments:

- the half-edges owned by v that intersect S_i;

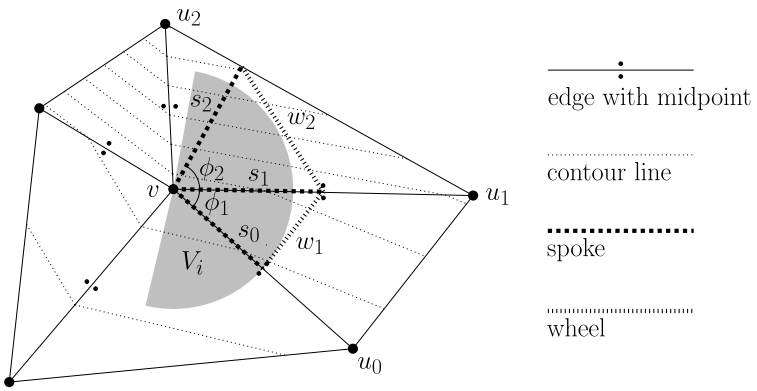

Fig. 6. The spokes of a sector S_i with neighborhood V_i (shaded) of v.

- for every triangle Δ that is incident to v and intersects S_i, the shortest line segment that connects v to a half-edge of Δ not owned by v.

See Fig. 6 for an example. Let $s_0, \ldots, s_k$ be the spokes of S_i (with duplicates removed) ordered in counterclockwise order around v. For $j \in \{1, \ldots, k\}$, let ϕ_j be the angle between s_{j-1} and s_j at v, and let the *wheel segment* w_j be the line segment that connects the other endpoints of s_{j-1} and s_j.

We claim that in any triangle Δ formed by any two consecutive spokes s_{j-1} and s_j and a wheel segment w_j, all angles are at least α. Let v, u_{j-1}, and u_j be the vertices of the terrain triangle that contains Δ in counterclockwise order. We now distinguish three cases: either both s_{j-1} and s_j are half-edges, or only s_{j-1} is a half-edge, or only s_j is a half-edge. In the first case (see Fig. 6, $j = 1$) Δ is isomorphic to $\triangle v u_{j-1} u_j$, which has minimum angle at least α. In the second case (see Fig. 6, $j = 2$) we have $2|s_{j-1}| \cos\phi_j = |s_j|$ while by the selection of spokes we have $|s_j| \leqslant |s_{j-1}|$, hence $\phi_j \geqslant \pi/3 \geqslant \alpha$. We have $\angle s_{j-1} w_j > \angle v u_{j-1} u_j \geqslant \alpha$ and by the sine rule we have $\sin\angle w_j s_j = |s_{j-1}| \sin\angle s_{j-1} w_j / |s_j| \geqslant \sin\alpha$, hence $\angle w_j s_j \geqslant \alpha$. The third case is symmetric to the second case, which completes the proof of our claim.

The sum of the angles $\phi_1, \ldots, \phi_k$ can be bounded as follows. Let the azimuth of a spoke be the non-negative angle it makes with the positive x-axis in the projection on the horizontal plane, and let the elevation of a spoke be the signed angle it makes with the horizontal plane. Without loss of generality, let the first spoke have azimuth 0. Since the local extrema in the slope profile of v separate neighborhoods, the elevations of $s_0, \ldots, s_k$ are in non-increasing or non-decreasing order. We assume they are in non-decreasing order; the other case is symmetric. Spherical geometry shows that the sum of azimuth and elevation difference of two vectors is an upper bound for their angle. Since both azimuth and elevation of the spokes are in non-decreasing order, this implies that $\sum_{j=1}^{k} \phi_j$ is bounded by the total azimuth range plus the total elevation range, so $\sum_{j=1}^{k} \phi_j \leqslant 3\pi$.

We now bound the length ratio of neighboring spokes, that is, the maximum of $|s_{j-1}|/|s_j|$, given ϕ_j. By the sine rule this is:

$$\begin{aligned}\max \frac{\sin\angle w_j s_j}{\sin\angle s_{j-1} w_j} &= \max \frac{\sin(\pi - \phi_j - \angle s_{j-1} w_j)}{\sin\angle s_{j-1} w_j} = \max\left(\frac{\sin\phi_j}{\tan\angle s_{j-1} w_j} + \cos\phi_j\right) \\ &= \sin\phi_j / \tan\alpha + \cos\phi_j.\end{aligned}$$

Hence the maximum ratio r of the length of any two spokes from $s_0, \ldots, s_k$ is:

$$r = \max_{\phi_1,\ldots,\phi_k} \prod_{i=1}^{k} (\sin\phi_j / \tan\alpha + \cos\phi_j) = \max_{\phi_1,\ldots,\phi_k} e^{\sum_{i=1}^{k} \ln(\sin\phi_j/\tan\alpha + \cos\phi_j)}.$$

Since the function $f(\phi) = \ln(\sin\phi/\tan\alpha + \cos\phi)$ is zero for $\phi = 0$ and concave on the domain $[0, \pi - 2\alpha]$, the maximum is attained when all angles ϕ_j are equal:

$$\begin{aligned} r &= \max_{k\in\{0,1,2,\ldots\};\alpha\leqslant\phi\leqslant 3\pi/k} e^{k\ln(\sin\phi/\tan\alpha+\cos\phi)} \\ &\leqslant \max_{k\in\mathbb{R};\alpha\leqslant\phi\leqslant 3\pi/k} e^{k\ln(\sin\phi/\tan\alpha+\cos\phi)} \\ &= e^{(3\pi/\alpha)\ln(\sin\alpha/\tan\alpha+\cos\alpha)} = (2\cos\alpha)^{3\pi/\alpha} < 2^{3\pi/\alpha}. \end{aligned}$$

The radius of V_i is the length of the shortest spoke, while the longest half-edge that intersects the neighborhood (if there is one) is the longest spoke. The length ratio of a head and its (full) edge is therefore at least half the length ratio of the shortest spoke and the longest spoke. This gives a lower bound of $1/2^{1+3\pi/\alpha}$. □

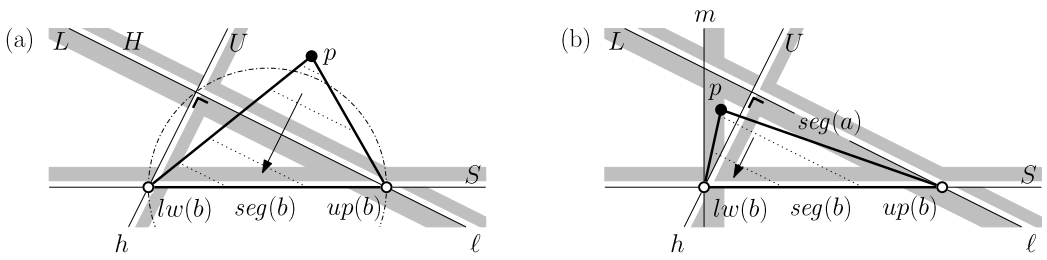

Fig. 7. Notation for the proof of Lemma 7.

Lemma 6. *If the angles in all triangles in a terrain are bounded from below by $\alpha > 0$, then a compliant subdivision of size $O(n/\alpha^2)$ exists.*

Proof. To bound the number of nodes we use the fact that each edge of the triangulation is cut into at most two heads and $2\lceil\frac{3}{2}\pi \ln 2/\alpha^2\rceil \approx 2\lceil 3.27/\alpha^2\rceil$ body segments. This follows from Lemma 3, setting $\rho = 1/2^{1+3\pi/\alpha}$ by Lemma 5. □

We get the following theorem:

Theorem 1. *A terrain that consists exclusively of triangles with minimum angle at least $\alpha > 0$, admits a compliant subdivision of size $O(n/\alpha^2)$. Consequently, it can be modeled by an acyclic descent graph of size $O(n/\alpha^2)$. Any down-path or up-path in such a terrain has $O(n/\alpha^2)$ vertices, and the total complexity of the river network or the strip map is $O(n^2/\alpha^2)$.*

3.3. Descent graphs for non-obtuse terrains

If the terrain $\mathcal{T}$ does not contain any obtuse triangles, we can construct the descent graph directly on the edges and vertices of $\mathcal{T}$, without further subdivision. The order relation $\succ$ is now simply defined by the elevations of the midpoints: $a \succ b$ if and only if $z(md(a)) > z(md(b))$.

Lemma 7. *If $\mathcal{G}_{\text{desc}}$ contains an arc from a to b, then $a \succ b$.*

Proof. Assume that there is an arc from a to b. If $seg(a)$ is the upper endpoint of the channel $seg(b)$ or if $seg(b)$ is the lower endpoint of the channel $seg(a)$, then $a \succ b$ since channels cannot be horizontal. It remains to discuss the case where there is a path of steepest descent from some point on $seg(a)$ to some point on $seg(b)$ across the interior of their common triangle Δ. First suppose that $seg(b)$ is an edge and $seg(a)$ is an edge or a vertex.

Let Γ be the plane supporting Δ. Let S be the open half-plane on Γ bounded by the line through $seg(b)$ and containing Δ. Let ℓ be the intersection of Γ with a horizontal plane through $up(b)$, and let h be the line on Γ orthogonal to ℓ through $lw(b)$. Note that ℓ is a contour line of Γ and the line h follows the direction of steepest descent on Γ. Let L and H be the closed lower and open higher half-planes of Γ bounded by ℓ, and let U be the open half-plane of Γ bounded by h and containing $seg(b)$. Let C be the open disk on Γ bounded by the smallest circumscribed circle of $seg(b)$. See Fig. 7(a) for an illustration. Let $p \in S$ be the vertex of Δ opposite of $seg(b)$. By Thales' theorem, since Δ is non-obtuse, p lies outside C, while C contains $L \cap U \cap S$. Hence p lies in H or in $L \setminus U$.

If p lies in H (Fig. 7(a)), then $z(p) > z(up(b)) \geqslant z(lw(b))$. If $seg(a) = p$, we have $z(md(a)) > z(md(b))$ and $a \succ b$. Otherwise $seg(a)$ is an edge of Δ incident to p, and so $up(a) = p$ and $lw(a)$ is either $lw(b)$ or $up(b)$. In both cases we have $z(md(a)) > z(md(b))$ and $a \succ b$.

If p lies in $L \setminus U$ (Fig. 7(b)), then $seg(a)$ must be the edge incident to p and $up(b)$ (since there is no path of steepest descent from the third edge or from p onto the interior of Δ). Since Δ does not have an obtuse angle at $lw(b)$, the vertex p lies in the closed half-plane of Γ that contains $seg(b)$ and is bounded by the line m through $lw(b)$ orthogonal to $seg(b)$. This implies that $z(p) > z(lw(b))$. From $up(a) = up(b)$ and $z(lw(a)) = z(p) > z(lw(b))$ we obtain $z(md(a)) > z(md(b))$ and $a \succ b$.

Symmetric arguments are applied if $seg(a)$ is an edge and $seg(b)$ is a vertex. □

Corollary 3. *$\mathcal{G}_{\text{desc}}$ is a planar directed acyclic graph.*

Proof. Since $\succ$ is a partial order, $\mathcal{G}_{\text{desc}}$ is acyclic. A planar straight-line embedding is given by placing every node v at the projection of $md(v)$ on a horizontal plane. □

We get the following theorem:

Theorem 2. *Any down-path or up-path in a terrain without obtuse triangles has $O(n)$ vertices, and the total complexity of the river network or the strip map is $O(n^2)$.*

4. Computing the river network and the watershed area map

This section shows how the acyclic descent graph can be used to construct the river network and the strip map of the terrain I/O-efficiently. Furthermore, we show how to compute, for every point on the river network, the area of its watershed. Recall that for points outside the river network the watershed area is zero; for these points one may be interested in the density of the flow, which we define and show how to compute.

4.1. Computing the descent graph

In Section 3 we showed that a terrain can be modeled by an acyclic descent graph which has $O(n/\alpha^2)$ nodes and arcs when the terrain is α-fat. We now show that this graph can also be computed in an I/O-efficient manner:

Lemma 8. *Given an α-fat terrain $\mathcal{T}$ of n triangles, we can compute an acyclic descent graph in $O(Sort(d))$ I/O's, where $d = O(n/\alpha^2)$ is the complexity of the descent graph.*

Proof. We describe how to compute a descent graph based on a compliant subdivision for a terrain that consists of triangles that are fat when measured in space. Our algorithm follows the construction described in Section 3.2. Computing a descent graph for a terrain without obtuse triangles (Section 3.3) is easier, and the algorithm described below can easily be simplified to do this.

We first sort the edges of the triangulation into a list that gives for each vertex first its own coordinates and then, consecutively, the coordinates of its neighbors in clockwise order. Thus each edge is represented twice in this list, once directed from one endpoint, and once directed from the other endpoint. Starting from any reasonable representation of the triangulation, this list can be produced in $O(Sort(n))$ I/O's.

We scan this list: for every vertex v, we scan its incident edges neighborhood by neighborhood; for each neighborhood we determine the radius of the neighborhood, cut every transfluent edge into two half-edges at its midpoint, and output the head and body segments of the half-edges incident to v. Each segment is output twice, once tagged with the coordinates of the triangle on its left, and once tagged with the coordinates of the triangle on its right. The complete list of segments is subsequently sorted by the tags. This takes $O(Sort(d))$ I/O's, where d is the number of segments.

To build the descent graph, we scan the list of segments sorted by triangle, and compute the arcs on each triangle in $O(Scan(d))$ I/O's. With each arc, we also store the (geometric) direction of steepest descent. For channels we also create arcs from the upper end to the channel, and from the channel to its lower end. This takes $O(Scan(n))$ I/O's.

In total, the descent graph is built in $O(Sort(d))$ I/O's, where d is the complexity of the descent graph. For α-fat terrains, $d = O(n/\alpha^2)$ by Theorem 1. □

4.2. Computing the river network

Recall that the river network consists of the channels in the terrain and the paths of steepest descent that start at the lower endpoints of channels. The channels can be extracted from the terrain in a straightforward way. The challenge in computing the rivers is tracing paths of steepest descent I/O-efficiently: we need to trace each such path from its upper endpoint across the triangles that it intersects, one segment at a time. We would like that a path that crosses k triangles can be computed in $O(Sort(k))$ I/O's. A general solution for this problem is not known in the I/O-model.

This is where the descent graph comes to the rescue. The key idea of our solution is that every segment of a path of steepest descent is captured by an arc in the descent graph. Thus, instead of tracing such paths on the original terrain, we trace them in the descent graph. Doing this path by path would not be any faster, since even for planar directed acyclic graphs it is not known how to preprocess them for fast path traversals. Instead, we trace *all* paths of steepest descent in parallel while traversing the descent graph in topological order (highest nodes first). In this way we can compute the segments of the paths of steepest descent I/O-efficiently in a batched way.

More precisely, our approach works as follows. We put the arcs of the descent graph $\mathcal{G}_{\text{desc}}$ in a list A sorted by $\succ$-order of their nodes of origin (see Section 3), with the nodes that appear first in $\succ$-order in front—ties broken in any well-defined way. Furthermore, we initialize an I/O-efficient priority queue Q that stores pairs of the type (v, p), where v is a node in the descent graph, and p is a point of $seg(v)$. The priority queue is also organized by $\succ$-order of the nodes of $\mathcal{G}_{\text{desc}}$—ties broken in the same way as above. Initially we fill Q with all pairs (v, p) where p is a bottom vertex of a channel, and v is the corresponding node in $\mathcal{G}_{\text{desc}}$.

We now repeat the following. From Q we extract the pair (v, p_1) with highest priority, and all further pairs $(v, p_2), (v, p_3), \ldots$ with the same priority. From the head of A we remove arcs $\overrightarrow{uw_1}$ until $u = v$ or until $v \succ u$. In the latter case, no arcs that lead out of v are found, and all paths of steepest descent that reach v end there—we proceed to extracting the next pair (v, p_1) from Q. Otherwise, there is at least one arc that leads out of v. We also remove any remaining arcs $\overrightarrow{vw_2}, \overrightarrow{vw_3}, \ldots$ that start at v from A. For each pair (v, p_i), we now select the arc $\overrightarrow{vw_j}$ that captures the path of steepest descent from p_i to a point q_i on $seg(w_j)$. If $seg(w_j)$ is not incident to p_i, we output the line segment from p_i to q_i as a segment of a river. If $seg(w_j)$ is a channel or the bottom vertex of a channel, the river that flows from

there is already being traced from there, otherwise we insert (w_j, q_i) into Q to ensure that the river will be traced further. After handling all pairs $(v, p_1), (v, p_2), \ldots$, we proceed by extracting the next pair from Q and repeat the above until Q is empty. We obtain the following.

Theorem 3. *Given a terrain $\mathcal{T}$ of n triangles and its acyclic descent graph, we can compute its river network in $O(Sort(r+d))$ I/O's, where r is the size of the river network and d is the size of the descent graph.*

Proof. We use the above algorithm. The algorithm is dominated by sorting $\mathcal{G}_{\text{desc}}$ (which takes $O(Sort(d))$ I/O's) and by the priority queue operations. With an I/O-efficient priority queue [6,8] these take $O(Sort(r))$ I/O's, since for every pair (v, p) inserted in Q, we have at least one vertex or segment in the river network. In theory, selecting the arc $\overrightarrow{vw_j}$ may be non-trivial if v has a very high out-degree. This can be solved by sorting the pairs $(v, p_1), (v, p_2), \ldots$ by increasing distance from $up(v)$ and sorting the arcs $\overrightarrow{vw_1}, \overrightarrow{vw_2}, \ldots$ by increasing distance from $up(v)$ to the projection of $md(w_i)$ onto $seg(v)$ in the direction of steepest ascent from $seg(w_i)$ to $seg(v)$. Thus we get that for any (v, p_i) with steepest descent leading to (w_j, q_i) and (v, p_k) with steepest descent leading to (w_l, q_l), we have that $k \geqslant i$ implies $l \geqslant j$. Summed over all nodes v, these sorting steps take $O(Sort(r))$ I/O's. Thus we get the river network in $O(Sort(r+d))$ I/O's. □

For an α-fat terrain we have that $d = O(n/\alpha^2)$ by Theorem 1, and computing $\mathcal{G}_{\text{desc}}$ takes $O(Sort(d))$ I/O's by Lemma 8.

Corollary 4. *Given an α-fat terrain of n triangles, we can compute its river network of size $r = O(n^2/\alpha^2)$ in $O(Sort(r + n/\alpha^2))$ I/O's.*

4.3. Computing the strip map and the watershed area map

Our algorithm for computing the river network can be extended to compute the subdivision of $\mathcal{T}$ induced by *all* channels, ridges, up-paths and down-paths starting from vertices of $\mathcal{T}$, that is, the strip map, in $O(Sort(s+d))$ I/O's, where s is the complexity of the strip map (and d is the complexity of the descent graph as above). The difference is that in this case we trace *all* down-paths, instead of just the down-paths that can be reached from channels. Furthermore, we do a second run of the algorithm upside-down to trace the up-paths, and some scanning and sorting passes to construct a representation of the strip map with all the necessary pointers between adjacent faces, edges and vertices. We also use our algorithm for tracing down-paths to construct, for every strip, a list of the triangles that intersect the strip.

The strip map can be used to compute the watershed areas of all points on the river network. Let q be a point in the terrain. We say that q lies at the foot of a strip σ if it lies in the interior of the channel segment at the foot of σ. The watershed $W(q)$ of q contains parts of the at most two strips that have q at their feet [25]; the area of each part is given by a quadratic function of the position of q, as shown below, and can be determined by scanning the list of triangles that intersect the strip. All other strips lie completely inside or outside the watershed of q; more precisely, a strip lies in $W(q)$ if and only if its lowest point lies at q or on the river network upstream of q.

Consider one of the strips containing q at their feet. Let $\triangle_1, \triangle_2, \ldots, \triangle_k$ be the triangles of this strip σ in order from the foot to the ridge (Fig. 8). Let e_1 be the channel at the foot of the strip, let e_i ($i \in \{2, \ldots, k\}$) be the edge between $\triangle_{i-1}$ and $\triangle_i$, and let e_{k+1} be the ridge at the top of the strip. For $i \in \{1, \ldots, k\}$, let t_i be the vertex common to e_i and e_{i+1}, let l_i be the vertex of σ on e_i which is furthest from t_i, and let u_i be the vertex of σ on e_{i+1} which is furthest from t_i. Now consider the up-path $q_1, \ldots, q_{k+1}$ that starts at $q_1 = q$, where q_i ($i \in \{1, \ldots, k+1\}$) is the intersection of the path with e_i. Because q_iq_{i+1} is parallel to l_iu_i, we have $|q_{i+1}t_i| = c_i \cdot |q_it_i|$, where $c_i = |t_iu_i|/|t_il_i|$, for $i \in \{1, \ldots, k\}$. If $t_{i+1} = t_i$, this implies $|q_{i+1}t_{i+1}| = c_i \cdot |q_it_i|$, otherwise $|q_{i+1}t_{i+1}| = |e_{i+1}| - c_i \cdot |q_it_i|$. In both cases $|q_{i+1}t_{i+1}|$ is a linear function of $|q_it_i|$, and by induction from $\triangle_1$ up to $\triangle_k$ it follows that $|q_it_i|$ is a linear function of $|qt_1|$ for any $i \in \{1, \ldots, k+1\}$.

Let r be the highest point at the foot of the strip. Similar to the up-path from q, let $r_1, \ldots, r_{k+1}$ be the up-path that starts from $r_1 = r$, where r_i ($i \in \{1, \ldots, k+1\}$) is the intersection of the path with e_i. The part of the strip that drains through q is bounded by the up-paths from q and r, and its area is given by $\sum_{i=1}^{k} |area(\triangle t_iq_iq_{i+1}) - area(\triangle t_ir_ir_{i+1})| = \sum_{i=1}^{k} c_i'(|q_it_i|^2 - |r_it_i|^2)$, where $c_i' = \pm area(\triangle t_il_iu_i)/|l_it_i|^2$ (the sign depends on the side of the strip where t_i is found). Since $|q_it_i|$ is a linear function of $|qt_1|$ and all other terms and factors are independent of q, it follows that the area of the part of the strip that drains through q is a quadratic function of the position of q on the foot of the strip (as specified by the distance of q to t_1). The coefficients of this function can be determined by scanning the triangles that intersect the strip from the foot up to the ridge.

To compute the piecewise quadratic function whose value is the watershed area for every point on the river network, we process the river network from the leaves to the root (that is, in downstream order) and collect, for every edge of the river network, the area functions for the strips to the left and to the right and the total area of the strips that drain into the subtree below (that is, upstream of) that edge. This can be done with standard techniques, for example with a post-order traversal of the river network. Recall that the watershed area is zero for any point outside the river network, and that the strip map has $O(n)$ faces, whose boundaries may divide the river network into $O(r+n)$ segments. We get the following:

Theorem 4. *Given a terrain $\mathcal{T}$ of n triangles and its acyclic descent graph, we can compute in $O(Sort(s+d))$ I/O's a piecewise quadratic function of $O(r+n)$ pieces whose value is the watershed area of every point in $\mathcal{T}$, where d is the size of the descent graph, r is the size of the river network and s is the size of the strip map of $\mathcal{T}$.*

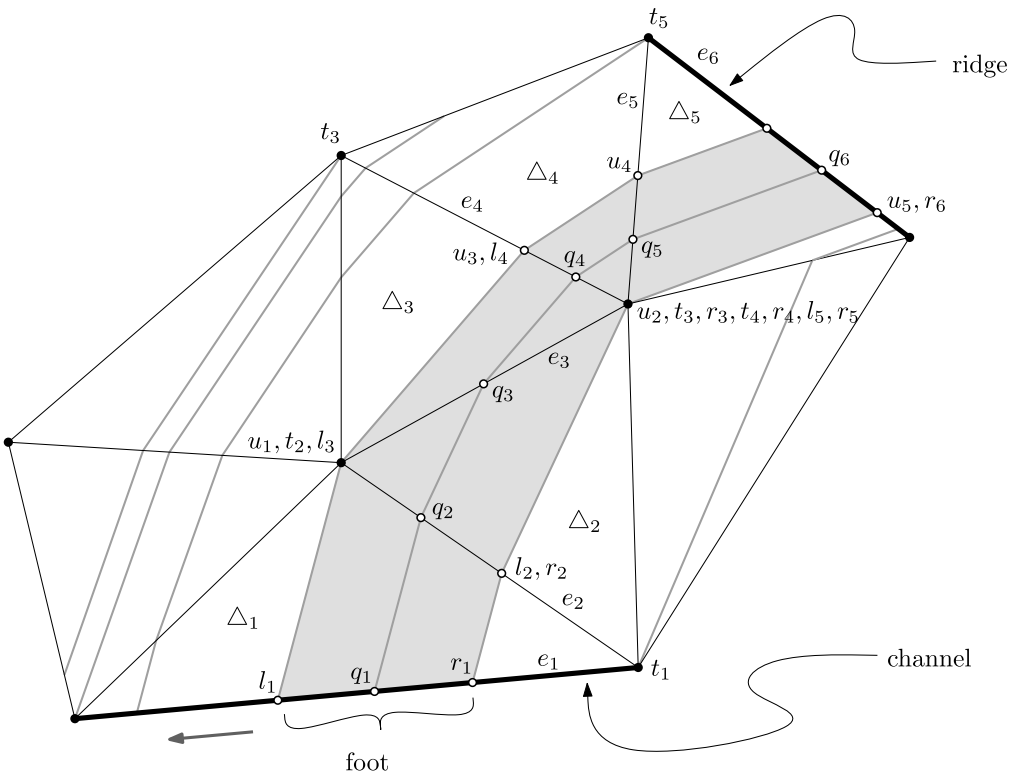

Fig. 8. Computing the watershed area of the point $q = q_1$ that lies at the foot of the shaded strip.

Corollary 5. *For an α-fat terrain $\mathcal{T}$ of n triangles we can compute in $O(Sort(s+n/\alpha^2))$ I/O's a piecewise quadratic function of $O(r+n)$ pieces whose value is the watershed area of every point in $\mathcal{T}$. The sizes r and s of the river network and strip map of $\mathcal{T}$ are $O(n^2/\alpha^2)$.*

4.4. Computing a flow density map

The watershed area of any point that is not on the river network is zero. For points outside the river network one may be interested in determining the *flow density*, that is, the amount of water that runs down a slope relative to the width of the slope; or in other words: how much water finds its way across how little space. Formally we can define the flow density of a point q as follows. Let s_ε be the intersection of the contour line(s) through q with a ball of diameter ε centered at q. Let $w_q(\varepsilon)$ be the area of the region from which water crosses s_ε. Then $\phi(q)$, the *flow density* of q, is defined as $\phi(q) := \lim_{\varepsilon\to 0} w_q(\varepsilon)/\varepsilon$. Note that the flow density is infinite for any point q that lies on the river network. A point q elsewhere in the terrain only receives water from the strip in which it lies, and within that strip, only from the triangle on which it lies and the triangles above. For simplicity we restrict our attention to points in the interior of strips. It is now easy to see that $\phi(q)$ is a linear function of the position of q on its triangle within its strip. Given the strip map, with the triangles crossed by each strip in order from the ridge to the channel, we can compute these functions strip by strip, and within each strip, triangle by triangle, in a single scan of the strip map. We get the following result:

Theorem 5. *Given a terrain $\mathcal{T}$ of n triangles and its acyclic descent graph, we can compute in $O(Sort(s+d))$ I/O's a piecewise linear function of $O(s)$ pieces whose value is the flow density of every point in $\mathcal{T}$, where d is the size of the descent graph and s is the size of the strip map of $\mathcal{T}$.*

5. A data structure for flow path and watershed queries

In this section we describe an I/O-efficient data structure for fast flow path and watershed boundary queries: given a point q on the terrain, we want to report the flow path starting at q and the boundary of the watershed of q. Recall that the watershed of q, which we denote by $W(q)$, is the region from which the water flows to q.

As explained in the previous section, the watershed of any point q on the river network can be found by traversing the subtree of the river network upstream of q, collecting the strips that drain into that subtree and parts of the (at most two) strips that have q at their feet. However, this would yield the watershed as a collection of strips, including strip boundaries

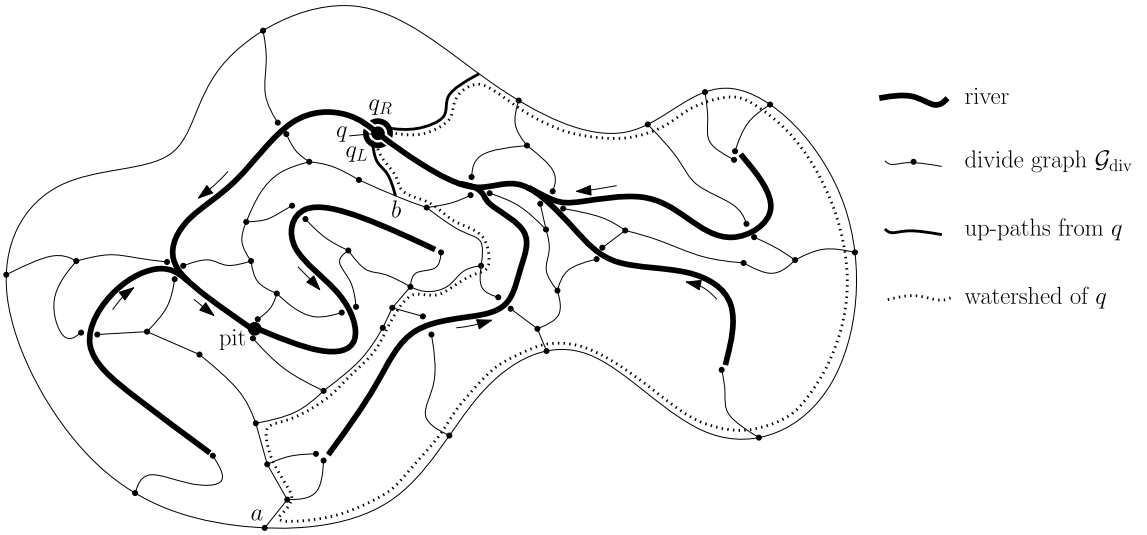

Fig. 9. Finding the watershed of q. We create two nodes q_R and q_L (shown as half-circles). We connect q_R to the existing graph $\mathcal{G}_{\text{div}}$ by drawing the up-path from q on the right shore of the river that contains q. We connect q_L to $\mathcal{G}_{\text{div}}$ by drawing the up-path on the left shore. When we now connect q_L and q_R to each other, the face of $\mathcal{G}_{\text{div}}$ that contains q is divided into two subfaces. The subface that lies upstream of q is the watershed of q.

that lie in the interior of the watershed. The total size of these boundaries may be much larger than the boundary of the complete watershed given as a simple polygon.

To address this problem McAllister [16] suggested to build a graph $\mathcal{G}_{\text{div}}$ (we will call it the *divide graph*) that consists of sections of ridges and certain up-paths in $\mathcal{T}$. The watershed boundary of a query point q can then be found by adding the up-paths from q to $\mathcal{G}_{\text{div}}$ and reporting the boundary of the face that lies upstream of q. This approach is illustrated in Fig. 9. This idea solves our problem only partially: First, we have to trace the up-paths from q I/O-efficiently. In the previous section, we managed to trace up-paths efficiently only when we had to trace many such paths simultaneously. Second, as illustrated in Fig. 9, many of the ridges and up-paths that define $\mathcal{G}_{\text{div}}$ lie in the interior of faces. These ridges and up-paths are there because they *might* become part of the boundary of the watershed when the up-paths from a certain query point are added—for example, this is the case with the path from a to b in Fig. 9 that is part of the boundary of $W(q)$. However, the same interior ridges and up-paths might be irrelevant for other queries. When following the boundary of $W(q)$ by simply following it clockwise and keeping to the right, we would waste time (and I/O's) traversing the subtrees of $\mathcal{G}_{\text{div}}$ that protrude from the watershed boundary into the watershed. Below we sketch how to refine $\mathcal{G}_{\text{div}}$ and store it face by face in a way that makes it possible to report watershed boundaries I/O-efficiently without going up and down dead ends.

5.1. The divide graph

Our divide graph $\mathcal{G}_{\text{div}}$ is defined as follows. For every vertex v of $\mathcal{T}$, and for every endpoint v of an up-path or a down-path from a vertex of $\mathcal{T}$, consider an infinitesimally small circle centered at the projection of v on the horizontal plane. Cut this circle where it is crossed by the projections of up-paths that start from v, channels that lead down to v, or the path of steepest descent from v. Note that we do not cut the circle at every down-path from v, but only at the channel or the down-path that descends steepest from v. Every piece of the circle that results constitutes a node of $\mathcal{G}_{\text{div}}$, which is not (directly) connected to the other pieces of the circle. The arcs of $\mathcal{G}_{\text{div}}$ correspond to (i) the ridges of $\mathcal{T}$ and (ii) two copies of each up- or down-path starting from a vertex v of $\mathcal{T}$. The two copies are assumed to lie at an infinitesimally small distance from the real course of the path, such that one copy runs to the left of the path, and the other to the right, leaving an infinitesimally narrow corridor between them.[6]

The arcs of $\mathcal{G}_{\text{div}}$ are connected to the nodes of $\mathcal{G}_{\text{div}}$ as follows. Every arc that corresponds to a path π that has v as an endpoint, is connected to the node whose piece of the circle around v is crossed by the projection of π on the plane. When an up- or down-path passes between two pieces of the circle on its way to or from v, the copy of the path that is offset to its left is connected to the piece of the circle to its left, and the copy that is offset to its right is connected to the piece of the circle to its right (see Fig. 10).

[6] This is necessary to deal with watersheds with degenerate boundaries: there may be watersheds with disconnected interiors where water flows from one component to another through a corridor of zero width. When we allow flow off the terrain, for example at a coast (Section 6 explains how to deal with that), similar degeneracies occur wherever a river reaches the coast through a corridor of zero width across the interior of a triangle rather than through a channel. To handle such cases we include parallel arcs that can bound such corridors on both sides.

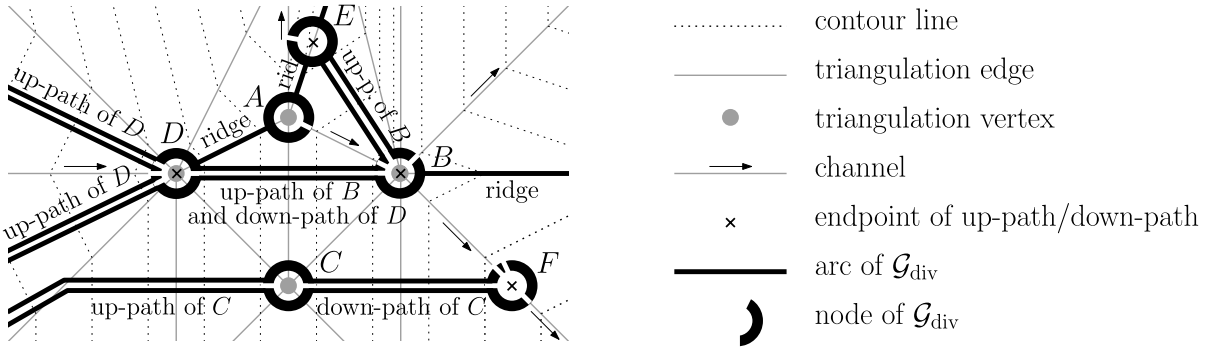

Fig. 10. An example of nodes and arcs in a divide graph.

To see that $\mathcal{G}_{\text{div}}$ can be used for watershed boundary queries, observe that $\mathcal{G}_{\text{div}}$ now contains all face boundaries of the strip map, except the channels. As observed in Section 4.3, the watershed of any query point q contains of a number of complete strips, plus parts of the at most two strips that have q at their feet—the only parts of the boundary of $W(q)$ that lie in the interior of strips are the up-paths from q. The rest of the boundary of $W(q)$ consists of face boundaries of the strip map. Channels are never on the boundary of a watershed, since they receive water from both adjacent triangles. Thus $\mathcal{G}_{\text{div}}$ contains all boundaries we need, except the up-paths from the query point which will have to be constructed at query time.

Streaming water never crosses the interior of a ridge or an up-path, since it always flows away from ridges and parallel to up-paths. Furthermore, at any point where ridges or up-paths meet, the corresponding arcs in $\mathcal{G}_{\text{div}}$ are incident to the same node—that is, the same piece of a circle—if and only if the geometric entities to which these arcs correspond are not separated by any path of steepest descent, that is, flowing water. Thus, no watershed is divided by face boundaries of $\mathcal{G}_{\text{div}}$. The projection of a vertex v on the horizontal plane always lies in the *interior* of a face of $\mathcal{G}_{\text{div}}$, and never on a node or arc of $\mathcal{G}_{\text{div}}$. Thus every vertex of $\mathcal{T}$ lies inside a well-defined watershed of a pit of the terrain.[7]

Using $\mathcal{G}_{\text{div}}$ we can find the watershed boundary of a point q as follows. If q is a pit, we report the boundary of the face $F(q)$ in $\mathcal{G}_{\text{div}}$ that contains q. Otherwise, we (conceptually) add the up-paths from q to $\mathcal{G}_{\text{div}}$. After this addition, let q_L and q_R be the nodes in $\mathcal{G}_{\text{div}}$ corresponding to the circular pieces around q immediately counterclockwise and clockwise, respectively, of the direction of steepest descent from q. The area $W(q)$ that drains through q is the area enclosed by the path in $\mathcal{G}_{\text{div}}$ that follows the boundary of $F(q)$ clockwise from q_R to q_L (see Fig. 9).

5.2. The data structure

We now sketch the data structure that makes it possible to trace the up-paths and the flow path (steepest-descent path) from q efficiently and to trace the boundary of $W(q)$ efficiently without going up and down arcs of $\mathcal{G}_{\text{div}}$ in the interior of $W(q)$. Our solution consists of four ingredients:

- the divide graph $\mathcal{G}_{\text{div}}$, stored face by face in a way that makes it possible to report watershed boundaries without going up and down dead ends (as explained below);
- the river network, preprocessed for fast downstream traversals (using the results of Hutchinson et al. [13]);
- the strip map, stored strip by strip to facilitate fast traversals of steepest-ascent and steepest-descent paths within each strip and to provide pointers into the divide graph and the river network;
- a point location structure on the strip map so that we can locate, for each query point q, the face (strip), the line segment, or the vertex of the strip map that contains it (using the results of Arge and Vahrenhold [5], Theorem 1).

5.2.1. Storing a face of $\mathcal{G}_{\text{div}}$

Let p be a pit. The boundary of the face $F(p)$ of $\mathcal{G}_{\text{div}}$ that represents the watershed of p can be seen as a clockwise cycle γ with trees protruding to its right, into the watershed of p.[8] Pick any node r that corresponds to a circular piece

[7] McAllister [16] uses a graph with fewer arcs: his graph models all ridges and the up-paths from a certain subset of the vertices. However, defining a subset of vertices with which the approach will work correctly is complicated. Moreover, simply adding *all* face boundaries of the strip map (except channel segments) has the advantage that the two up-paths that need to be traced at query time are restricted to lie within a single strip each. This simplifies the design of an I/O-efficient data structure. Since we add our arcs in such a way that they never intersect the path of flowing water, our addition does no harm to the correctness of the query algorithm.

[8] If one pit's watershed encloses another pit's watershed, the divide graph contains a path that connects the outer boundary of the outer watershed to the boundary of the inner watershed. This path forms the divide in the outer watershed that separates water that flows clockwise around the inner

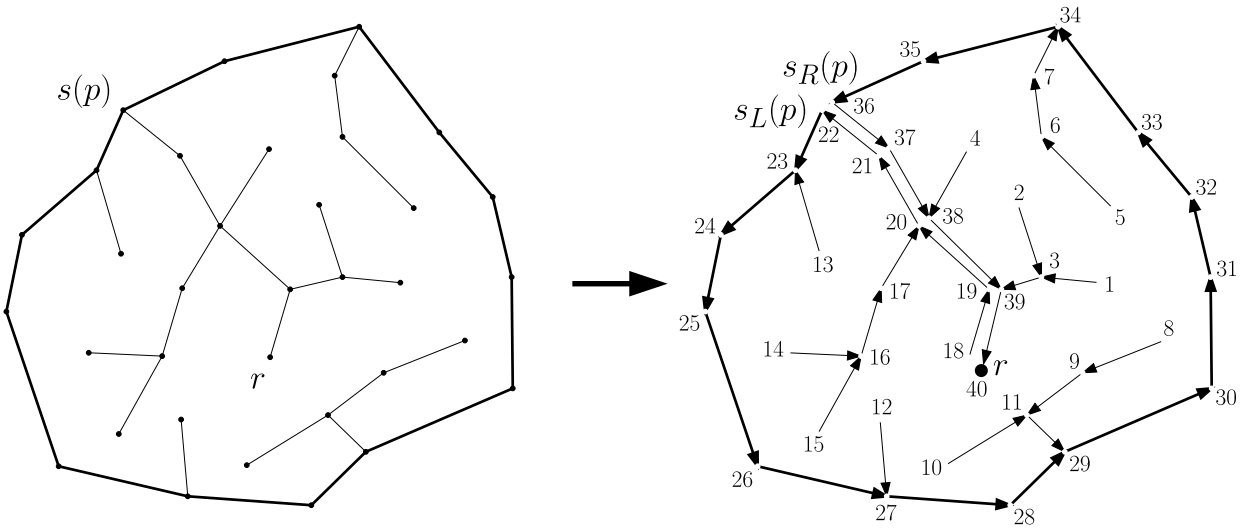

Fig. 11. Transforming the boundary of a face into a rooted tree. The fat edges form the boundary cycle γ. In the right figure, the edges are directed from child to parent. The ranks of the vertices in a postorder traversal of the tree are given.

around p; and let π be the path from r to a node $s(p)$ of the boundary γ, such that $s(p)$ is the first and only node of γ on π. Our idea is to "cut" the graph at r, and convert the boundary of $F(p)$ to a tree (see Fig. 11). To do this, we start by splitting all edges and vertices on π into a left copy and a right copy. Every right vertex is connected to the right edges incident to it, and to any trees that protrude into the watershed of p to the right of π. Furthermore, the right copy $s_R(p)$ of $s(p)$ is connected to the edge that follows $s(p)$ in γ. Every left vertex is connected to the left edges incident to it, and to any trees that protrude into the watershed of p to the left of π. Furthermore, the left copy $s_L(p)$ of $s(p)$ is connected to the edge that precedes $s(p)$ in γ. We root the resulting tree at the right copy of r. We break up arcs that represent up-paths or down-paths in segments: one segment for each triangle crossed by the path, with vertices at the points where the path crosses a transfluent edge of the triangulation. Let the resulting tree be $\mathcal{B}(p)$; we store with each node its rank in a postorder traversal of the tree, and preprocess the tree for fast leaf-to-root traversals with the method of Hutchinson et al. [13].

The divide graph $\mathcal{G}_{\text{div}}$ may contain several nodes per ridge (one for each up-path that ends on that ridge). To make sure that we do not report more nodes than necessary when a watershed boundary follows a ridge, we may modify the tree structure of a face boundary as follows. For any node v, let the *highest collinear ancestor* u of v be the ancestor of v such that all nodes on the path from u to v (including u and v) lie on the same ridge. If $u \neq v$, we make u the parent of v. These shortcuts will enable us to skip over vertices of $\mathcal{G}_{\text{div}}$ on the interior of ridges if they are not corners of the reported watershed boundary.

5.2.2. Storing the faces of the strip map

The strip map consists of strips, line segments between strips (which are segments of up-paths, ridges or channels), and vertices. We store the strip map using one of the standard topological representations of a planar subdivision so that we can access for each face (strip) its neighbor faces, the boundary lines segments and the vertices between them. In addition to this we store with each strip: (i) the triangles that cross the strip, in the order in which they appear in any down-path from the ridge to the channel; (ii) a pointer to the highest-ranked node of $\mathcal{B}$ that lies on the ridge section at the top of the strip, where $\mathcal{B}$ is the tree that represents the boundary of the watershed that contains the strip; (iii) a pointer to the place where the lowest point of the strip is stored in the river network. With each segment of a channel we store pointers to the strips on each side, and a pointer to the lower endpoint of the edge that contains the segment in the river network. With a pit p we store pointers to $s_L(p)$ and $s_R(p)$.

5.3. The query algorithms

Lemma 9. *The watershed boundary of a query point q can be reported with $O(l(s) + k/B)$ I/O's, where $l(s) = O(\log_B^2 n)$ is the number* I/O*s needed for planar point location in a strip map of size s and k is the complexity of the watershed boundary reported.*

watershed from water that flows counterclockwise around the inner watershed. In such cases we treat the outer watershed's boundary as a cycle that follows the outer boundary clockwise to the divide, then goes down the divide to the inner boundary, counterclockwise around the inner watershed, back up the divide, and then clockwise along the outer boundary.

Proof. To report the watershed boundary of a query point q, we first need to locate q in the strip map. We may use the point location structure by Arge and Vahrenhold [5], which can be built in $O((s/B)\log_B s)$ I/O's and answers queries in $l(s) = O(\log_B^2 s)$ I/O's. The way to proceed then depends on where q is found: in the interior of a strip, on a ridge, on an up-path or down-path between strips, in a pit, or elsewhere on the river network. We discuss two cases that illustrate all the necessary techniques; the reader may verify that the other cases are variations thereof.

If q is in the interior of a strip, the watershed of q consists of the up-path from q. We find this path by first retrieving the triangles crossed from the list stored with the strip, from the ridge down to the triangle that contains q, in $O(1+k/B)$ I/O's. We then trace the up-path from q across these triangles, working our way back up from q to the ridge, in $O(1+k/B)$ I/O's.

If q is found in the interior of a segment of a channel, its watershed is found by conceptually adding the up-paths from q to the divide graph $\mathcal{G}_{\text{div}}$, and then reporting the path that connects q_R and q_L in $\mathcal{G}_{\text{div}}$ and encloses the area upstream of q. This path consists of the up-paths from q and the boundary of $F(q)$ clockwise from the upper end of the right up-path to the upper end of the left up-path. The triangulation edges crossed by the up-paths from q to $\mathcal{G}_{\text{div}}$ are stored in the strips on each side of the channel, and they can be retrieved in $O(1+k/B)$ I/O's. Let $pit(q)$ denote the pit in the face that contains q. From the strips we also retrieve the highest-ranked nodes u_L and u_R in $\mathcal{B}(pit(q))$ on the ridge sections at the top of the left and the right strips, respectively. Since q is not a pit, its watershed does not include the pit of its face, and the part of the boundary of $F(q)$ clockwise from u_R to u_L never passes through the root of $\mathcal{B}(pit(q))$. Therefore it can be found by tracing it in $\mathcal{B}(pit(q))$ from u_R up to the lowest common ancestor with u_L—which is the lowest ancestor v of u_R that has a postorder rank higher than that of u_L—and then from u_L up to that same node v. This takes $O(1+k/B)$ I/O's with the data structure of Hutchinson et al. To report the boundary of the watershed of q, we put the up-path from q right of the channel, the path from u_R to v, the path from u_L to v (in reverse order), and the up-path from q left of the channel (in reverse order) together in $O(k/B)$ I/O's. □

Lemma 10. *The flow path (path of steepest descent) from a query point q can be reported with $O(l+k/B)$ I/O's, where $l(s) = O(\log_B^2 n)$ is the number I/O's needed for planar point location in a strip map of size s and k is the complexity of the flow path reported.*

Proof. Flow path queries starting at a point q can be answered by tracing the steepest-descent path from q. To do this we first need to locate q in the strip map. If q is in the interior of a strip, then we first trace the steepest downslope path from q in the strip down to the channel at the foot of the strip; this can be done by retrieving the triangles stored in the strip in $O(1+k/B)$ I/O's. From the foot of the strip the path continues on the river network: this part of the path can be traced with $O(1+k/B)$ I/O's from the river network tree, which is pre-processed for fast downstream traversals. If q is on the river network, then we only need to trace the path from q down in the river network tree as above. Overall this uses $O(l(s)+k/B)$ I/O's, where $l(s) = O(\log_B^2 n)$ is the number I/O's needed for planar point location and k is the complexity of the flow path. □

5.4. Building the data structure

Let us denote by $q(s)$ and $l(s)$ the number of I/O's needed to build and query, respectively, a planar point location structure on a map of size s. Using the results of Arge and Vahrenhold [5] we have $q(s) = O((s/B)\log_B s)$ and $l(s) = O(\log_B^2 s)$.

Theorem 6. *A data structure of size $O(s)$ for answering flow path and watershed boundary queries on a fat terrain $\mathcal{T}$ can be computed in $O(q(s)+Sort(s)+Sort(d))$ I/O's, where d is the size of the descent graph and s is the size of the strip map of $\mathcal{T}$. The structure reports the watershed boundary or the flow path of any query point q in $O(l(s)+k/B)$ I/O's, where k is the complexity of the answer.*

Proof. The descent graph can be built in $O(Sort(d))$ I/O's using Lemma 8, where d is the size of the descent graph. Using the algorithm of Theorem 3 and variations thereof, we compute the edges of the strip map and the arcs of the divide graph in $O(Sort(s+d))$ I/O's. In the same process we can also construct the lists of triangles crossed by each strip.

Next we link the arcs of the divide graph together in clockwise order around each face; list-rank them to sort them by face; transform each face's boundary into a rooted tree with post-order numbering; traverse the trees to add shortcuts over collinear ancestors; and preprocess the trees for fast leaf-to-root traversals with the method of Hutchinson et al. [13]. These steps can all be done using standard techniques in $O(Sort(s))$ I/O's.

The data structure that stores the river network is built in $O(Sort(r))$ I/O's [13], and the point location structure on the strip map is built in $O((s/B)\log_B(s/B))$ I/O's [5]. The necessary pointers from the strip map into the divide graph and the river network can be added in $O(1)$ scanning and sorting passes in $O(Sort(s))$ I/O's.

In total, the data structure is built in $O(q(s)+Sort(s)+Sort(d))$ I/O's. For α-fat terrains, $d = O(n/\alpha^2)$ by Theorem 1. □

6. Dealing with degeneracies and flow off the terrain

In Section 2 we made a number of assumptions about the terrain, so that we could explain our algorithms, data structures and proofs without having to deal with "degenerate" cases. Some of our assumptions may be violated by real-life inputs, and we discuss briefly how to deal with that.

6.1. Steepest descent is not unique, flat areas

We made the assumption that the direction of steepest descent is unique for any point on the terrain that is not a pit. This assumption can be violated for a number of reasons, the most obvious being the existence of flat areas, that is, horizontal triangles or channels.

First, if the triangles incident to a ridge have the same slope, then this violates the assumption but actually causes no problem in the algorithm.

Second, if a vertex has more than one direction of steepest descent, we need to decide which of the possible outgoing edges of the descent graph the flow is going to take. This decision simply cannot be made based on the terrain data alone. Once the decision has been made it can be incorporated in the computation easily.

The third and final case is that the terrain contains a flat area. If such a flat area has at least one *spill point*—a vertex with a lower neighbor—water that flows into the flat area should find its way to the spill point.

We formally define a flat area as a maximal connected set of triangles and channels that are all horizontal at the same elevation. To enable our algorithm to deal with such a flat area, we can compute a breadth-first search[9] numbering of the vertices of the area, starting from one or more *spill points*: vertices that have a lower neighbor. We then conceptually raise the elevation of each vertex in the flat area by a small amount proportional to its rank in the breadth-first-search order. This guarantees that the modified flat area has no horizontal edges and no local minima other than the spill points. Thus, from each point in the raised flat area, the path of steepest descent leads down to a spill point. The perturbation can be done purely symbolically, and its effect is to assign a direction of flow to horizontal channels and triangles. The direction of flow on neighboring non-horizontal triangles remains unchanged.

The approach just described is just one way of making sure that water that enters a flat area is routed to its spill points. Thus the area draining to the flat area is correctly included in the spill point's watershed, and contributes to the watershed area of all points downstream of the spill point. Whether the flow network on the flat area itself corresponds to the flow network on the real terrain cannot be determined based on the assumption that water flows downhill alone, and the same could be said about how the incoming water would be divided over multiple spill points. Other heuristics to make a realistic guess of the flow network in a flat area could be considered, see for example Danner et al. [10] for results with grid terrains.

6.2. Water that flows off the terrain

To prevent degeneracies on the boundary of the terrain, we assumed that no water flows off the terrain. Thus all edges on the boundary of the terrain can be treated as ridges. We can deal with terrains where water flows off the boundary as follows. Let the *inner terrain* be the terrain for which we actually have data, and let the *inner boundary* be its boundary. We surround this terrain by a deep trench with one pit. The trench is surrounded by a wall of ridges that form the outer boundary of the terrain. Thus all water that flows off the inner terrain is caught in the trench, and we do not have to deal with water flowing off the terrain. Since no water from outside the inner terrain can flow (back) into the inner terrain, the addition of the trench does not change the watersheds of points in the inner terrain.

6.3. Edges that are parallel to the steepest descent on an adjacent triangle

If edges may be parallel to the steepest descent on an adjacent triangle, we have to deal with several special cases. We can do so in the style of McAllister [16]. McAllister discusses the case where a local maximum in the slope profile of a vertex v lies on the interior of a triangle Δ incident to v—that is, not on an edge. McAllister treats such a local maximum as two parallel ridges on the interior of the triangle with an infinitesimally narrow corridor between them. The ridges separate the water that reaches v across Δ from the water that flows down on Δ to the left of the steepest-descent path to v, and from the water that flows down on Δ to the right of the steepest-descent path to v (note that we take the same approach when we add two copies of every up-path to $\mathcal{G}_{\text{div}}$ in Section 5). We can deal with edges that are parallel to the steepest descent on one or both adjacent triangles in a similar manner:

- edges that are parallel to the direction of steepest descent on both adjacent triangles can be handled as a pair of ridges as described above;
- edges that receive water from one adjacent triangle and are parallel to the direction of steepest descent on the other adjacent triangle Δ can be treated as a channel, with just next to it a ridge that separates the channel from the interior of Δ;

[9] Theoretically I/O-efficient techniques for breadth-first search in planar graphs are available [15].

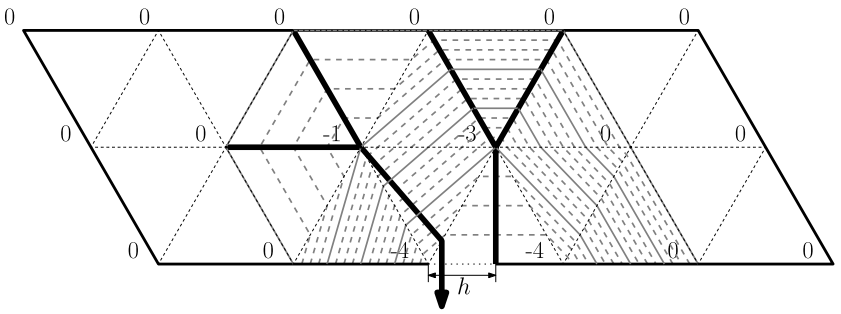

Fig. 12. A suitable terrain of 20 triangles with degree one. Fat lines indicate rivers; numbers indicate elevations. The middle edge of the bottom side is horizontal, and across the half of that edge that is closest to the center of the terrain, one river flows off the terrain. One more river flows off the terrain, at the right end of the door h, but this is irrelevant for our construction.

- ridges that are parallel to the direction of steepest descent on one adjacent triangle Δ can be treated as any other ridge (moving it an infinitesimally small distance left or right would not make any difference for flow and watershed computations, since no water runs down along the edge itself).

7. Lower bounds on the worst-case complexity of river networks

In this section we describe tight lower bounds for the worst-case complexity of river networks in terrains that consist of only fat triangles, only non-obtuse triangles, or only Delaunay triangles.

In particular, in Section 7.1 we describe how to construct a terrain such that the projection on a horizontal plane consists of n equilateral unit triangles and the river network has complexity $\Omega(n^2)$. By scaling the construction in vertical direction, the angles inside the triangles measured in space can be made to lie between $(60-\varepsilon)^\circ$ and $(60+\varepsilon)^\circ$ for any $\varepsilon > 0$. This shows that the $O(n^2)$ bound from Section 3 on the complexity of the river network of a fat or non-obtuse terrain, is tight in the worst case, regardless whether angles are measured in space or in the projection on a horizontal plane.

In Section 7.2 we describe how to construct a terrain of n triangles such that the projection on a horizontal plane is a Delaunay triangulation and the river network has complexity $\Omega(n^3)$. This shows that the upper bound of $O(n^3)$ on the worst-case complexity of general terrains [11] is tight also for Delaunay triangulations.

7.1. A lower-bound construction for fat, non-obtuse terrains

We describe a terrain, represented by a regular grid of n equilateral unit triangles, that has a river network with $\Theta(n^2)$ vertices. Our construction uses parallelogram-shaped terrains that have the following property: one side of the parallelogram has a horizontal middle edge, and one or more rivers flow off the terrain across the interior of the half h of that edge that is closest to the center of the terrain—see Fig. 12 for an example. We call terrains of this form *suitable terrains*, h the *door*, and the number of rivers that flow off the terrain across h the *degree* of the terrain. All vertices on the boundary of the terrain have height 0, except the endpoints of the door, which are some distance (the *depth*) below 0. The terrain shown in Fig. 12 is the basic ingredient of our construction. It is a suitable terrain of degree one with sides of five and two units, respectively, so that it contains 20 triangles in total.

We can now construct suitable terrains of larger degree recursively. Let $T(0)$ be the terrain just described, with length $l(0)=5$, width $w(0)=2$, depth $d(0)=4$, and degree $g(0)=1$. For $k \in \{1,2,3,\ldots\}$ we construct a terrain $T(k)$ such that the sides parallel to the door have length $l(k)=2w(k-1)+3$, the other sides have length $w(k)=l(k-1)$, and the terrain has degree $g(k) \geqslant 2g(k-1)$. The construction is shown in Fig. 13: we take two copies of $T(k-1)$, one turned such that the door is on the right and one turned such that the door is on the left, and we connect these copies such that all rivers that flow through the doors of the copies of $T(k-1)$ leave $T(k)$ as separate rivers through the door of $T(k)$.

By induction one can now prove $g(k) \geqslant 2^k$, and $l(k) = 8\cdot 2^{k/2} - 3$ when k is even, while $l(k) = 5\cdot 2^{(k+1)/2} - 3$ when k is odd. The number of triangles in the terrain is $2l(k)w(k) = 2l(k)l(k-1) < 80\cdot 2^k$ for all $k \in \{0,1,2,\ldots\}$.

Theorem 7. *There is a terrain of n triangles that are equilateral in the projection on a horizontal plane, with a river network of $\Theta(n^2)$ vertices.*

Proof. We start with $T(\lfloor \log(n/160) \rfloor)$: this terrain consists of less than $n/2$ triangles and $\Omega(n)$ rivers flow out of it across a single edge z on its boundary. We extend the terrain with a meandering valley of $n/2$ triangles such that all $\Omega(n)$ rivers that run across z cross $\Theta(n)$ edges of the valley floor while remaining disjoint, see Fig. 14. Thus we get a terrain of n triangles with a river network of complexity $\Omega(n^2)$. Since all triangles are non-obtuse, it follows from Theorem 2 that the complexity of the river network is $\Theta(n^2)$. □

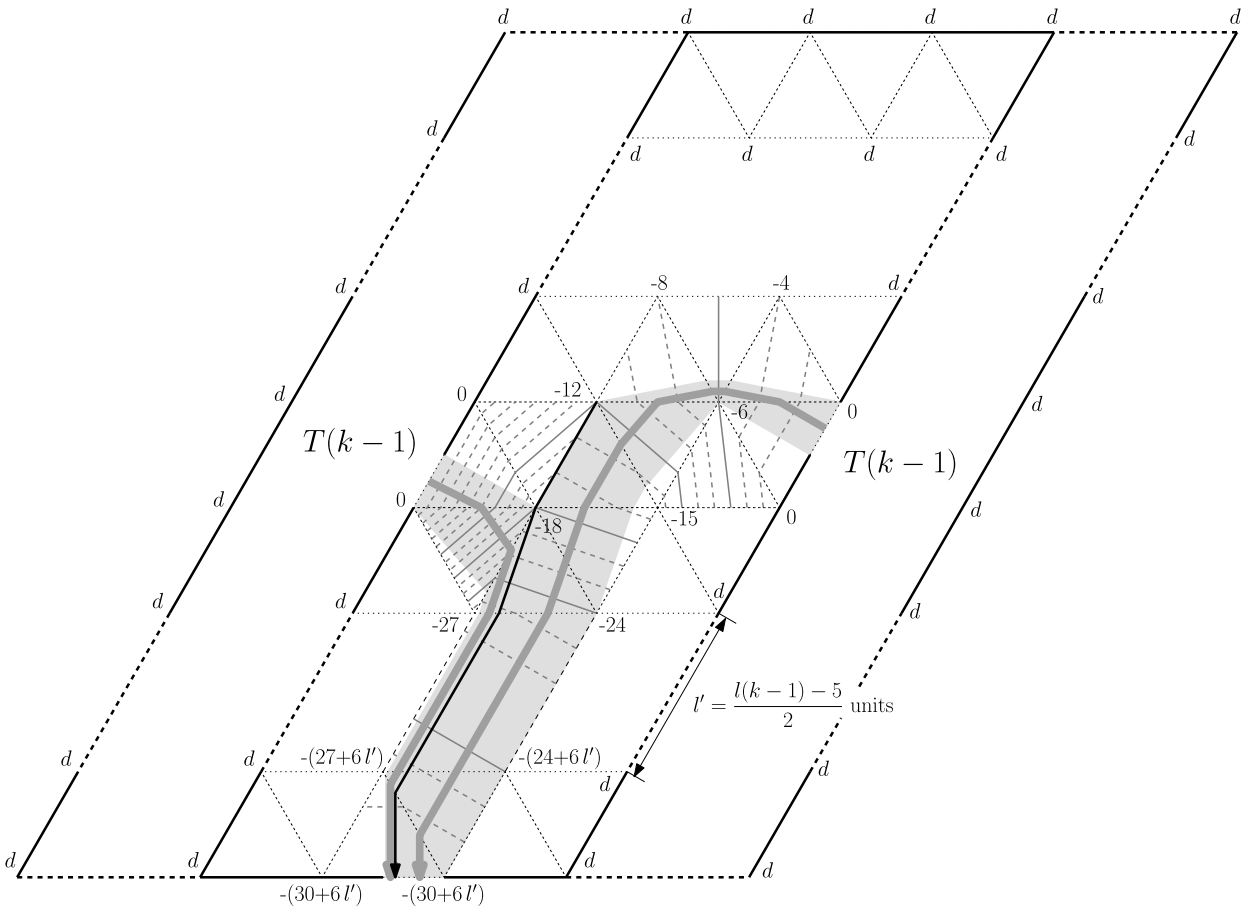

Fig. 13. $T(k)$ is constructed from two copies of $T(k-1)$ and a strip of length $l(k-1)$ and width 3 in between. The fat gray lines indicate bundles of separate rivers; numbers indicate elevations relative to the door of $T(k-1)$, where d is the depth of $T(k-1)$. All rivers that enter the strip through the doors of the copies of $T(k-1)$ are routed to the door of $T(k)$ without merging; this is accomplished by carefully choosing the elevations of the vertices inside the strip (and thus, the directions of steepest descent on the triangles in the strip) as indicated in the figure.

7.2. A lower-bound construction for Delaunay-triangulated terrains

In this section we describe a terrain such that the projection on a horizontal plane is a Delaunay triangulation of n vertices and the river network has complexity $\Theta(n^3)$. The terrain resembles an octagonal pyramid, which is designed such that rivers that start near the top flow down on a counterclockwise spiraling course around the top, completing $\Theta(n)$ cycles before flowing off the terrain.

A rough sketch of the construction is given in Fig. 15. To induce the counterclockwise course of the rivers, the four southern faces of the pyramid are inclined towards the counterclockwise direction. Thus these faces get successively steeper in counterclockwise direction. The four northern faces are inclined and get successively steeper in *clockwise* direction. To nevertheless induce a *counterclockwise* course of the rivers on the northern faces, these faces resemble stairways. The steps of these stairways are almost parallel to the iso-contour lines of the faces, with a slight inclination towards the counterclockwise direction. Thus rivers running on these steps turn counterclockwise around the top.

Our construction includes $\Theta(n)$ rivers that each complete $\Theta(n)$ cycles around the top, crossing $\Theta(n)$ edges on one of the southern faces on each cycle. Below we describe the construction in detail. For given k, l, and m, we make a terrain that has l disjoint rivers that each cross at least k edges m times. We first define the $23+k+4l+220m$ vertices of this terrain and how they are triangulated. Then we explain the spiraling course of the rivers.

7.2.1. The northern faces

The northern faces $\mathcal{F}_i$, $i \in \{0,1,2,3\}$ cover the areas in the directions $i\pi/4$ to $(i+1)\pi/4$, respectively, from the top of the pyramid. These faces contain $19+220m$ vertices whose x- and y-coordinates are as follows (see Fig. 16):

$$O = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

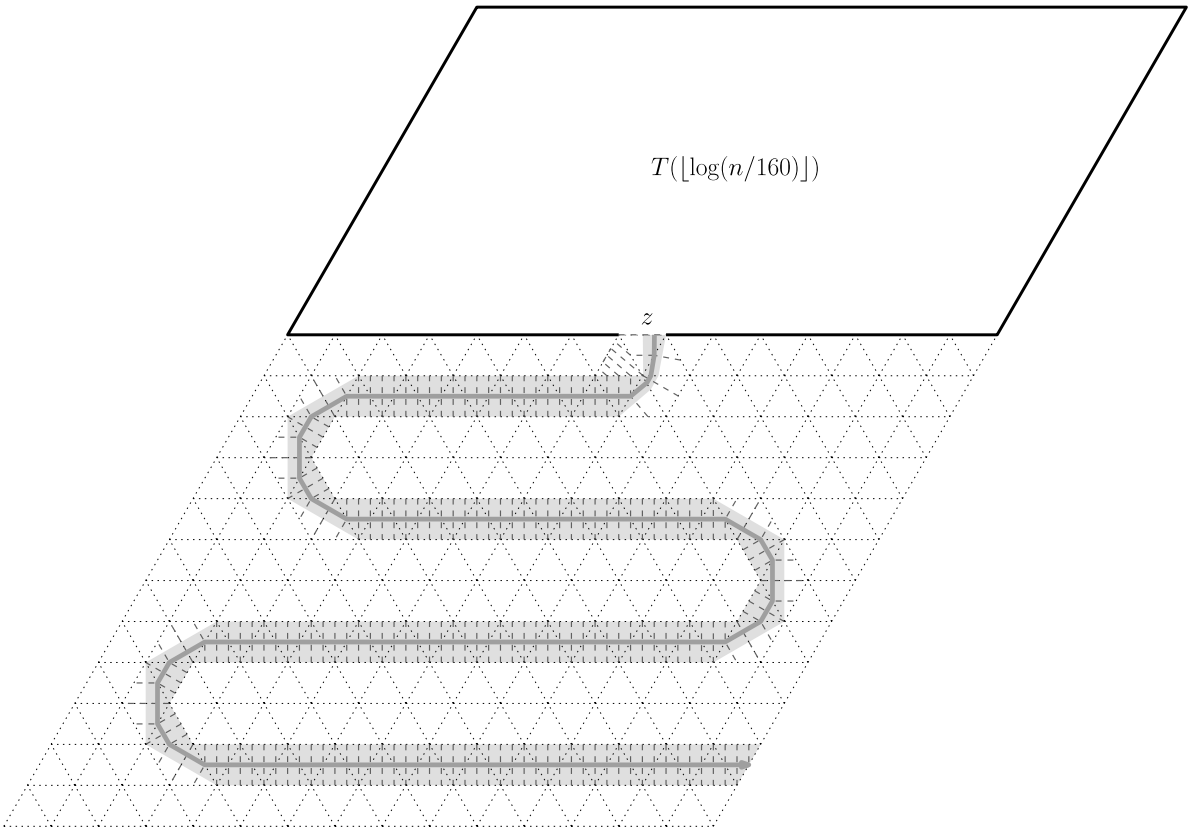

Fig. 14. A terrain with a river network of complexity $\Omega(n^2)$. The shaded area indicates a bundle of $\Theta(n)$ rivers, emerging from $T(\lfloor \log(n/160) \rfloor)$, that never merge while crossing $\Theta(n)$ edges in a long serpentine-shaped valley.

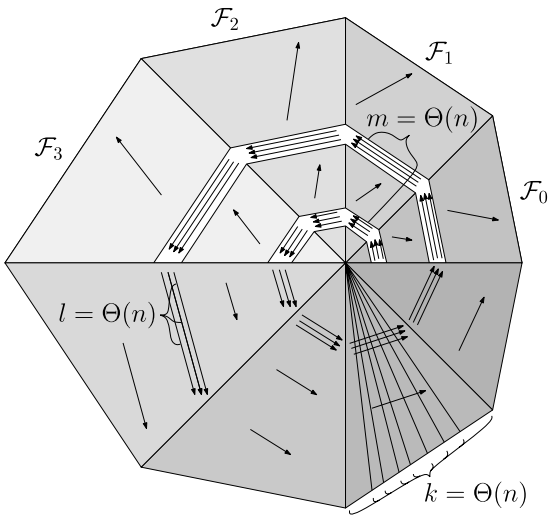

Fig. 15. A rough sketch of the lower-bound construction for Delaunay triangulations.

$$A_{i,j} = \left(\frac{7\sqrt{2}}{32}\right)^{4-i} \frac{2^{2j}}{2^{20m+1}} \begin{pmatrix} \cos\frac{\pi}{4}i \\ \sin\frac{\pi}{4}i \end{pmatrix} \quad \text{for } i \in \{0,1,\ldots,4\},\ j \in \{0,1,\ldots,10m\},$$

$$B_{i,j} = A_{i,j} + \frac{63}{337}(A_{i+1,j} - A_{i,j}) \quad \text{for } i \in \{0,1,\ldots,3\},\ j \in \{0,1,\ldots,10m\},$$

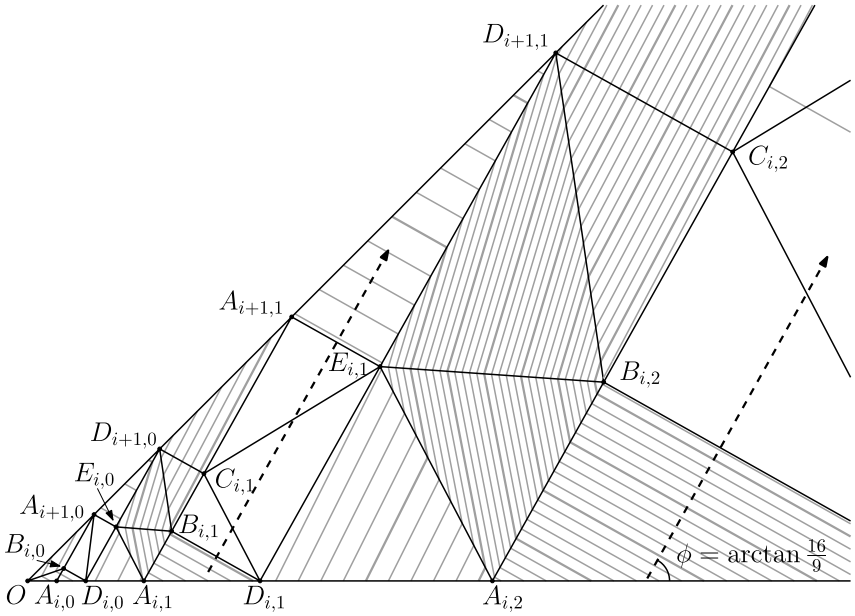

Fig. 16. The relative positions of the vertices $O, A_{i,j}, B_{i,j}, C_{i,j}, D_{i,j}, E_{i,j}$ of the northern face $\mathcal{F}_i$, their triangulation, and contour lines. The dashed arrows indicate the direction of rivers flowing on the "steps" formed by the quadrangles $A_{i,j}D_{i,j}D_{i+1,j}A_{i+1,j}$.

$$C_{i,j} = A_{i,j} + \frac{137}{337}(A_{i+1,j} - A_{i,j}) \quad \text{for } i \in \{0,1,\ldots,3\},\ j \in \{1,2,\ldots,10m\},$$

$$D_{i,j} = \left(\frac{7\sqrt{2}}{32}\right)^{4-i} \frac{2^{2j+1}}{2^{20m+1}} \begin{pmatrix} \cos\frac{\pi}{4}i \\ \sin\frac{\pi}{4}i \end{pmatrix} \quad \text{for } i \in \{0,1,\ldots,4\},\ j \in \{0,1,\ldots,10m\},$$

$$E_{i,j} = D_{i,j} + \frac{137}{337}(D_{i+1,j} - D_{i,j}) \quad \text{for } i \in \{0,1,\ldots,3\},\ j \in \{0,1,\ldots,10m\}.$$

Each face $\mathcal{F}_i$ is triangulated by the triangles $\triangle O A_{i,0}B_{i,0}$, $\triangle O B_{i,0}A_{i+1,0}$, $\triangle A_{i,0}D_{i,0}B_{i,0}$, $\triangle B_{i,0}D_{i,0}A_{i+1,0}$, $\triangle A_{i+1,0}D_{i,0}E_{i,0}$, and $\triangle A_{i+1,0}E_{i,0}D_{i+1,0}$, and, for $j \in \{1,2,\ldots,10m\}$, $\triangle D_{i,j-1}A_{i,j}E_{i,j-1}$, $\triangle E_{i,j-1}A_{i,j}B_{i,j}$, $\triangle E_{i,j-1}B_{i,j}D_{i+1,j-1}$, $\triangle D_{i+1,j-1}B_{i,j}C_{i,j}$, $\triangle D_{i+1,j-1}C_{i,j}A_{i+1,j}$, $\triangle A_{i,j}D_{i,j}B_{i,j}$, $\triangle B_{i,j}D_{i,j}C_{i,j}$, $\triangle C_{i,j}D_{i,j}E_{i,j}$, $\triangle C_{i,j}E_{i,j}A_{i+1,j}$, and $\triangle A_{i+1,j}E_{i,j}D_{i+1,j}$.

The elevations of the above-mentioned vertices are as follows:

$$\begin{aligned}
&\mathit{height}(O) = 0,\\
&\mathit{height}(A_{i,j}) = -2^{2j}(192/91)^i,\\
&\mathit{height}(B_{i,j}) = -2^{2j+1}(192/91)^i,\\
&\mathit{height}(D_{i,j}) = -2^{2j+1}(192/91)^i,\\
&\mathit{height}(E_{i,j}) = -2^{2j}(192/91)^{i+1}.
\end{aligned}$$

The elevations of the vertices $C_{i,j}$ are chosen so that every vertex $C_{i,j}$ is placed on the line through $B_{i,j}$ and $A_{i+1,j}$.

Observe that we have $\mathit{height}(A_{i,j}) > \mathit{height}(B_{i,j}) = \mathit{height}(D_{i,j}) > \mathit{height}(A_{i+1,j}) = \mathit{height}(E_{i,j}) > \mathit{height}(D_{i+1,j})$. Hence in the quadrangle $A_{i,j}D_{i,j}D_{i+1,j}A_{i+1,j}$ all contour lines are parallel to $\overline{B_{i,j}D_{i,j}}$. Thus, in the vertical projection on a horizontal plane, all water flows parallel to the direction of $\overline{A_{i,j}A_{i+1,j}}$, which makes an angle of $\phi = \arctan\frac{16}{9}$ with $\overline{O D_{i,10m}}$.

7.2.2. Creating rivers

To create l distinct rivers, we remove the edge $D_{0,0}A_{1,0}$ from $\mathcal{F}_0$ and refine the terrain in the rectangle $B_{0,0}D_{0,0}E_{0,0}A_{1,0}$ with $2+4l$ additional vertices as follows.

Consider the vertical projection of the terrain on a horizontal plane. First we draw two circles, one is the circumcircle of the triangle $\triangle A_{0,0}D_{0,0}B_{0,0}$ and the other is the circumcircle of the triangle $\triangle A_{1,0}E_{0,0}D_{1,0}$—see Fig. 17(a). Let the points U_{2l} on the first circle and V_{2l} on the latter circle be the endpoints of the shortest line segment between these two circles. Let $\mathcal{C}$ be the circle that has $\overline{U_{2l}V_{2l}}$ as a diameter. Let U_0 and V_0 be the intersection points of $\mathcal{C}$ and the circle through O, $B_{0,0}$ and $A_{1,0}$, where U_0 is the intersection closest to $B_{0,0}$. We place $U_1, U_2, \ldots, U_{2l-1}$ on $\mathcal{C}$ in counterclockwise order between U_0 and U_{2l}, and we place $V_1, V_2, \ldots, V_{2l-1}$ on $\mathcal{C}$ in clockwise order between V_0 and V_{2l}, such that $\overline{U_iV_i}$ is parallel to $\overline{U_{2l}V_{2l}}$ for all $i \in \{1,\ldots,2l\}$.

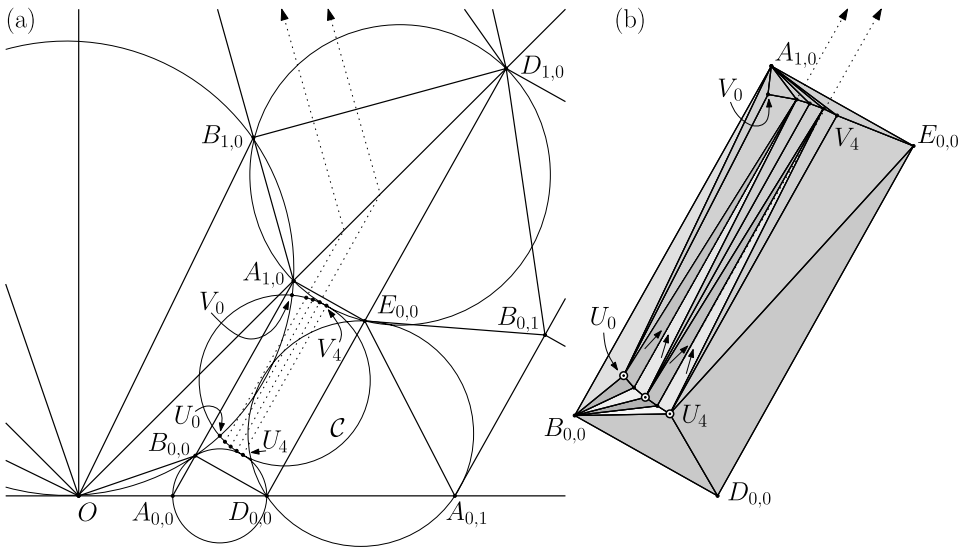

Fig. 17. (a) How the points U_i, V_i are positioned (in this example, $l = 2$). (b) How the rectangle $B_{0,0}D_{0,0}E_{0,0}A_{1,0}$ is triangulated to make l distinct rivers. The marked vertices (U_0, U_2, U_4) are raised above the plane that contains the rectangle $B_{0,0}D_{0,0}E_{0,0}A_{1,0}$. For odd i, this tilts the triangles adjacent to the edge U_iV_i towards the edge, thus creating channels that start rivers in the direction of the arrows.

We triangulate the rectangle $B_{0,0}D_{0,0}E_{0,0}A_{1,0}$ by means of the triangles $\triangle B_{0,0}D_{0,0}U_{2l}$, $\triangle U_{2l}D_{0,0}E_{0,0}$, $\triangle U_{2l}E_{0,0}V_{2l}$, $\triangle B_{0,0}U_0A_{1,0}$, $\triangle A_{1,0}U_0V_0$, and $\triangle A_{1,0}V_{2l}E_{0,0}$, and for all $j \in \{1,\ldots,l\}$ the triangles $\triangle B_{0,0}U_{2j-1}U_{2j-2}$, $\triangle B_{0,0}U_{2j}U_{2j-1}$, $\triangle U_{2j-2}V_{2j-1}V_{2j-2}$, $\triangle U_{2j-2}U_{2j-1}V_{2j-1}$, $\triangle U_{2j-1}U_{2j}V_{2j-1}$, $\triangle U_{2j}V_{2j}V_{2j-1}$, $\triangle A_{1,0}V_{2j-2}V_{2j-1}$, and $\triangle A_{1,0}V_{2j-1}V_{2j}$—see Fig. 17(b).

Initially we set the heights of all vertices U_i and V_i for $i \in \{0,1,\ldots,2l\}$ so that these vertices are all coplanar with the rectangle $B_{0,0}D_{0,0}E_{0,0}A_{1,0}$. After that, we raise the vertices U_{2j} for $j \in \{0,\ldots,l\}$. Thus the edges $\overline{U_{2j-1}V_{2j-1}}$, for $j \in \{1,\ldots,l\}$, become channels, each collecting a non-zero amount of water from the two adjacent triangles $\triangle U_{2j-2}U_{2j-1}V_{2j-1}$ and $\triangle U_{2j-1}U_{2j}V_{2j-1}$.

7.2.3. The southern faces

To define the southern faces of the pyramid, we define $2+k$ more vertices:

$$R = \begin{pmatrix} -1 \\ -1 \end{pmatrix},$$

$$S_i = \begin{pmatrix} \sin\frac{\pi i}{4k} \\ -1-\cos\frac{\pi i}{4k} \end{pmatrix} \quad \text{for } i \in \{0,1,\ldots,k-1\},$$

$$T = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

The location of the vertices of the southern faces and their triangulation are illustrated in Fig. 18.

Considering the southern faces of the pyramid in counterclockwise order, the first face is triangulated by $\triangle OA_{4,0}R$, $\triangle A_{4,0}D_{4,0}R$, and, for $j \in \{1,2,\ldots,10m\}$, $\triangle D_{4,j-1}A_{4,j}R$ and $\triangle A_{4,j}D_{4,j}R$. Observe that all points $D_{4,j}$ and $A_{4,j}$ lie on the line segment $\overline{OD_{4,10m}}$ in three-dimensional space, and hence all triangles on this face are coplanar.

The second face consists of the triangle $\triangle ORS_0$.

The third face consists of the triangles $\triangle OS_{i-1}S_i$, for all $i \in \{1,2,\ldots,k-1\}$, and the triangle $\triangle OS_{k-1}T$.

The fourth face is triangulated by $\triangle A_{0,0}OT$, $\triangle D_{0,0}A_{0,0}T$, and, for $j \in \{1,2,\ldots,10m\}$, $\triangle A_{0,j}D_{0,j-1}T$ and $\triangle D_{0,j}A_{0,j}T$. Similar to the first face, all triangles on this face are coplanar.

The heights of the vertices on the southern faces are set as follows:

$$height(R) = \frac{13}{6}height(D_{4,10m}),$$

$$height(S_0) = \frac{13}{6}height(R),$$

$$height(T) = \frac{13}{12}height(S_0).$$

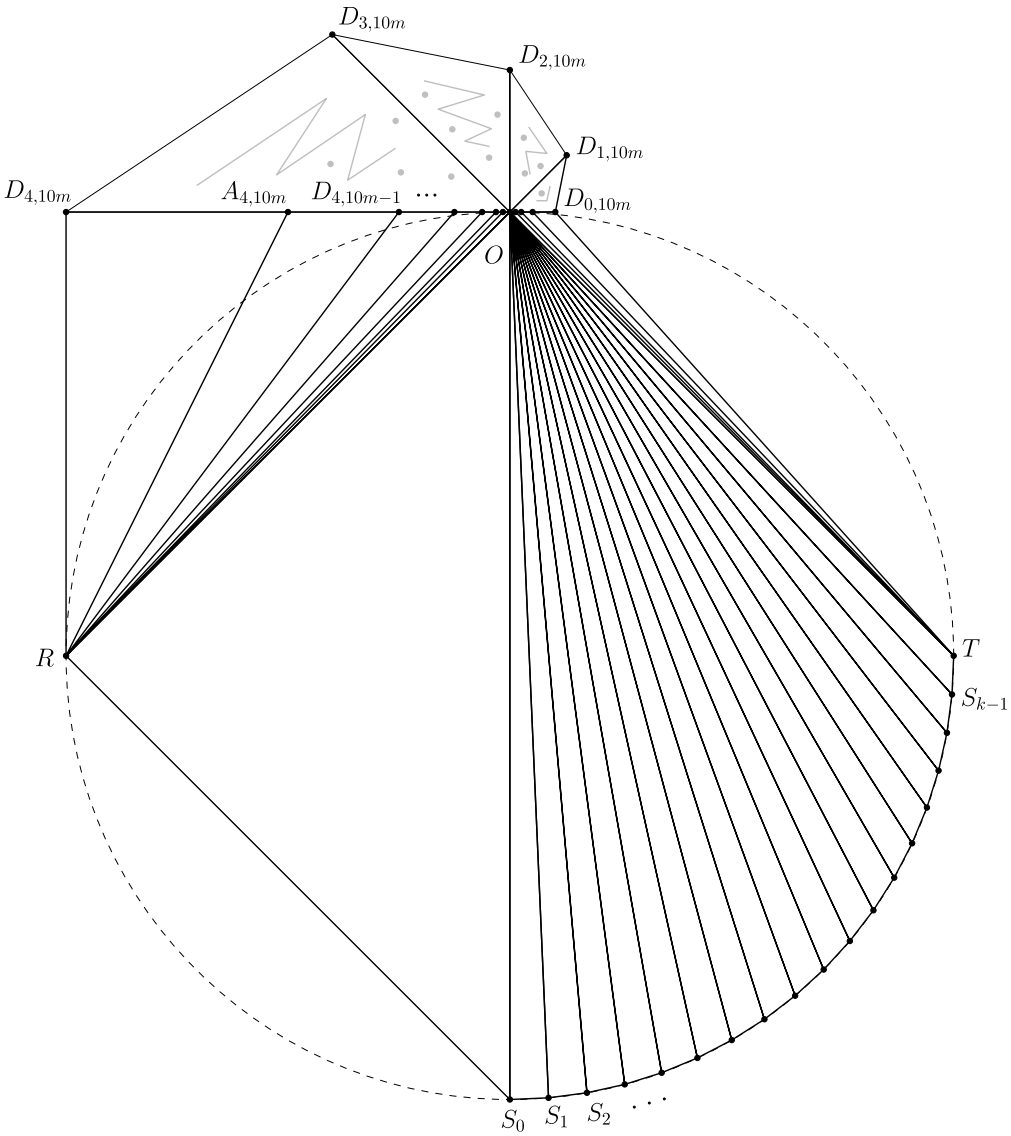

Fig. 18. The vertices and the triangulation of the southern faces.

The heights of $S_1, S_2, \ldots, S_{k-1}$ are set so that all these vertices lie on the plane containing O, S_0, and T in three-dimensional space.

Considering the line segments $\overline{OD_{4,10m}}$, $\overline{OR}$, $\overline{OS_0}$, $\overline{OT}$, $\overline{OD_{0,10m}}$ that bound the southern faces of the pyramid, we observe that the slope of each line segment is $\frac{13}{12}\sqrt{2}$ times the slope of the previous line segment. Hence, on each southern face all water flows in the direction that makes an angle $\theta = \arctan((\frac{13}{12}\sqrt{2} - \cos(\frac{\pi}{4}))/\sin(\frac{\pi}{4})) = \arctan\frac{7}{6}$ with the line segment that bounds the face on the clockwise side.

7.2.4. The course of rivers

By the above observations any river that crosses the interior of an edge $\overline{A_{i,j}D_{i,j}}$ (for $i \in \{0,1,2,3\}$, $j \in \{0,1,\ldots,10m\}$ with the possible exception of $\overline{A_{0,0}D_{0,0}}$) flows to $\overline{A_{i+1,j}D_{i+1,j}}$, without merging with other rivers; on this course the horizontal distance of the river to O increases by a factor $\sqrt{2}/(1 - 1/\tan\phi) = \frac{16}{7}\sqrt{2}$. On each of the southern faces, any river that enters it on the clockwise side flows towards the counterclockwise side without merging with other rivers, and the horizontal distance to O increases by a factor $\sqrt{2}/(1 - 1/\tan\theta) = 7\sqrt{2}$ (unless this causes the river to flow off the terrain). Fig. 19 illustrates how the rivers flow around O: on the northern faces of the pyramid the river (solid line) and the clockwise edge of the face make an angle of ϕ. On the southern faces the river (dotted line) and the clockwise edge make an angle of θ.

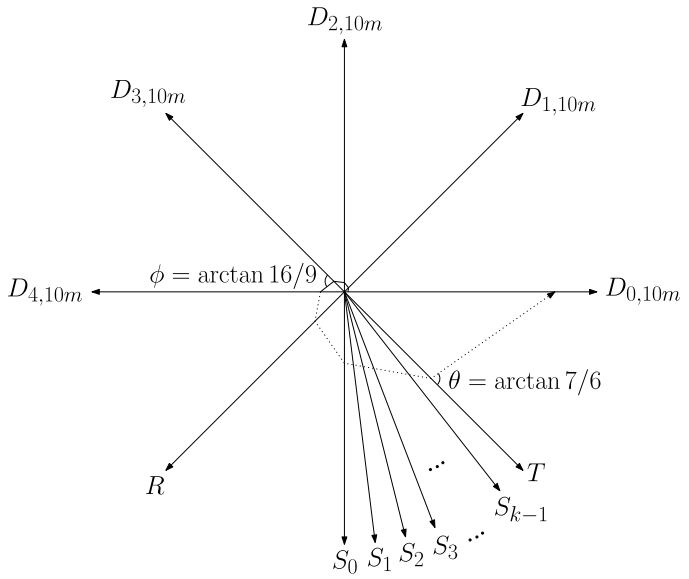

Fig. 19. The river spirals out and gets further from the top of the pyramid. On every cycle of the spiral, the river crosses at least k edges.

As a result, any river that crosses the interior of an edge $\overline{A_{1,j}D_{1,j}}$ (for $j \in \{0, 1, \ldots, 10m - 10\}$) makes a full cycle around the top of the pyramid, increasing the horizontal distance to O by a factor $(\frac{16}{7}\sqrt{2})^4(7\sqrt{2})^4 = 2^{20}$, crossing all edges $\overline{OS_0}, \overline{OS_1}, \ldots, \overline{OS_{k-1}}$ on the way and eventually crossing the interior of $\overline{A_{1,j+10}D_{1,j+10}}$ (note that $A_{1,j+10} = 2^{20}A_{1,j}$ and $D_{1,j+10} = 2^{20}D_{1,j}$).

All l rivers that start at the vertices $U_1, U_3, \ldots, U_{2l-1}$ cross the interior of $A_{1,0}D_{1,0}$, and thus complete m cycles, each time crossing more than k edges, before they cross $A_{1,10m}D_{1,10m}$, after which they soon flow off the terrain. It can easily be verified that the triangulation described above constitutes a Delaunay triangulation.

The complete terrain has $23 + k + 4l + 220m$ vertices. We choose $k = l = m = (n - 23)/225$, and thus we have proved the following:

Theorem 8. *For any n that is big enough, there is a triangulated terrain of n vertices, such that the vertical projection of the terrain on the horizontal plane forms a Delaunay triangulation, and the river network of the terrain has $\Omega(n^3)$ vertices.*

8. Discussion

The algorithms described in this paper are based on the assumption that water that flows over the terrain always flows downhill in the direction of steepest descent, regardless of its volume or inertia. As a result the terrain is subdivided into areas that drain to the same local minimum. However, real-life data can have many artificial local minima as a result of sampling errors or artifacts of the triangulation. This may lead to fragmentation of rivers and watersheds that are connected in reality. For practical application of algorithms such as described in this paper, preprocessing is needed to eliminate local minima that appear to be artifacts of sampling and modeling. Recently, Agarwal et al. [2] published an I/O-efficient algorithm that does this by *flooding*, that is, raising the terrain around a local minimum until it lies high enough to drain over a saddle point, the *spill point*, into a lower minimum's watershed. This results in a terrain with flat (horizontal) areas, which can be handled as in Section 6.

The main problem that remains seems to be the complexity of the river network and the strip map. De Berg et al. [11] proved that this is $\Theta(n^3)$ in the worst case, and asked if better bounds could be proven for Delaunay triangulations or other triangulations with well-shaped triangles. We showed that the $\Theta(n^3)$ bound still holds for Delaunay triangulations, but triangulations with only fat or only non-obtuse triangles have river networks and strip maps of size $\Theta(n^2)$ in the worst case.

In practice, a complexity of $\Theta(n^2)$ would not be manageable. However, our lower-bound constructions are still contrived: they involve bundles of $\Theta(n)$ disjoint rivers that together cross $\Theta(n)$ triangles without ever merging. In practice one might hope that such rivers would soon merge into one big river in a channel. Indeed Yu et al. [25] observed that the river network seems to have linear complexity in practice. It would be interesting to investigate this more thoroughly and to quantify under which conditions this is the case: our lower bound example shows that familiar conditions on the shapes of the triangles do not suffice.

Our algorithms for the I/O-efficient construction of river networks and watershed query data structures rely on the fatness of the terrain, but in a relatively robust way: local deviations from the fatness assumption are not punished too badly. Only

in the direct neighborhood of small angles will many nodes be put in the descent graph. To make our algorithms run most efficiently, one might want to preprocess the terrain to eliminate small and/or obtuse angles, so that the descent graph remains small. Of course this would change the shape of the surface of the terrain. Since the triangulated terrain is only an approximation of the real terrain, this is not necessarily a problem. However, research would be needed to determine under what conditions and how a terrain could be preprocessed I/O-efficiently such that the triangulated terrain after preprocessing is an equally acceptable approximation of the real terrain as the triangulated terrain before preprocessing. Alternatively, one may look into algorithms that construct triangulated terrains from raw elevation samples in such a way that the triangulated surface is a good approximation of the real terrain and small and/or obtuse angles are avoided at the same time.

Acknowledgements

The authors thank Frank van der Stappen and Hee-Kap Ahn for their contribution in the early stages of the paper.

References

[1] P.K. Agarwal, L. Arge, T.M. Murali, K. Varadarajan, J.S. Vitter, I/O-efficient algorithms for contour line extraction and planar graph blocking, in: Proc. ACM–SIAM Symposium on Discrete Algorithms, 1998, pp. 117–126.
[2] Pankaj Agarwal, Lars Arge, Kevin Yi, I/O-efficient batched union-find and its applications to terrain analysis, in: Proc. ACM Symposium on Computational Geometry, 2006, pp. 167–176.
[3] A. Aggarwal, J.S. Vitter, The input/output complexity of sorting and related problems, Communications of the ACM 31 (9) (1988) 1116–1127.
[4] L. Arge, J. Chase, P. Halpin, L. Toma, D. Urban, J.S. Vitter, R. Wickremesinghe, Flow computation on massive grid terrains, GeoInformatica 7 (4) (2003) 104–128. Earlier version appeared in: Proc. 10th ACM International Symposium on Advances in Geographic Information Systems (ACM–GIS'01).
[5] L. Arge, J. Vahrenhold, I/O-efficient dynamic planar point location, Computational Geometry 29 (2004) 147–162.
[6] Lars Arge, The buffer tree: A technique for designing batched external data structures, Algorithmica 37 (1) (2003) 1–24.
[7] Lars Arge, Andrew Danner, Sha-Mayn Teh, I/O-efficient point location using persistent *b*-trees, J. Exp. Algorithmics 8 (2003) 1.2.
[8] G.S. Brodal, J. Katajainen, Worst-case efficient external-memory priority queues, in: Proc. Scandinavian Workshop on Algorithms Theory, in: LNCS, vol. 1432, 1998, pp. 107–118.
[9] Y.-J. Chiang, M.T. Goodrich, E.F. Grove, R. Tamassia, D.E. Vengroff, J.S. Vitter, External-memory graph algorithms, in: Proc. ACM–SIAM Symposium on Discrete Algorithms, 1995, pp. 139–149.
[10] A. Danner, T. Mølhave, K. Yi, P.K. Agarwal, L. Arge, H. Mitasova, Terrastream: From elevation data to watershed hierarchies, in: Proc. ACM Symposium on Geographic Information Systems, 2007, pp. 212–219.
[11] M. de Berg, P. Bose, K. Dobrint, M. van Kreveld, M. Overmars, M. de Groot, T. Roos, J. Snoeyink, S. Yu, The complexity of rivers in triangulated terrains, in: Proc. Canadian Conference on Computational Geometry, 1996, pp. 325–330.
[12] Michael T. Goodrich, Jyh-Jong Tsay, Darren E. Vengroff, Jeffrey Scott Vitter, External-memory computational geometry, in: Proceedings of the 34th Annual Symposium on Foundations of Computer Science, 1993, pp. 714–723.
[13] D. Hutchinson, A. Maheshwari, N. Zeh, An external-memory data structure for shortest path queries, in: Proc. Annual Combinatorics and Computing Conference, in: LNCS, vol. 1627, 1999, pp. 51–60.
[14] N.L. Jones, S.G. Wright, D.R. Maidment, Watershed delineation with triangle-based terrain models, Journal of Hydraulic Engineering 116 (10) (1990) 1232–1252.
[15] Anil Maheshwari, Norbert Zeh, I/O-optimal algorithms for planar graphs using separators, in: Proc. ACM–SIAM Symposium on Discrete Algorithms, 2002, pp. 372–381.
[16] M. McAllister, The computational geometry of hydrology data in geographic information systems, PhD thesis, Univ. of British Columbia, 1999.
[17] M. McAllister, A watershed algorithm for triangulated terrains, in: Proc. Canadian Conference on Computational Geometry, 1999.
[18] Michael McAllister, Jack Snoeyink, Extracting consistent watersheds from digital river and elevation data, in: Ann. Conf. American Society for Photogrammetry and Remote Sensing, 1999.
[19] Esther Moet, Marc van Kreveld, A. Frank van der Stappen, On realistic terrains, Computational Geometry 41 (2008) 48–67.
[20] O. Palacios-Velez, W. Gandoy-Bernasconi, B. Cuevas-Renaud, Geometric analysis of surface runoff and the computation order of unit elements in distributed hydrological models, Journal of Hydrology 211 (1998) 266–274.
[21] A.T. Silfer, G.J. Kinn, J.M. Hassett, A geographic information system utilizing the triangulated irregular network as a basis for hydrologic modeling, in: Proc. Auto-Carto 8, 1987, pp. 129–136.
[22] D.M. Theobald, M.F. Goodchild, Artifacts of TIN-based surface flow modeling, in: Proc. GIS/LIS'90, 1990, pp. 955–964.
[23] G. Tucker, S. Lancaster, N. Gasparini, S. Rybarczyk, An object-oriented framework for hydrology geomorphic modeling using triangulated irregular networks, Computers and Geosciences 27 (8) (2001) 959–973.
[24] M. van Kreveld, Digital elevation models TIN algorithms, in: Algorithmic Foundations of Geographic Information Systems, in: LNCS, vol. 1340, Springer-Verlag, 1997, pp. 37–78.
[25] S. Yu, M. van Kreveld, J. Snoeyink, Drainage queries on TINs: From local to global and back again, in: Proc. 7th Int. Symp. on Spatial Data Handling, 1996, pp. 13A.1–13A.14.